2004

Development of a human visual system with the ability to detect inconsistent events

Jeremiah D. Brazeau

Development of a Human Visual System with the Ability to Detect Inconsistent Events

Jeremiah D. Brazeau

Department of Computer Science
Rochester Institute of Technology

A thesis, submitted to the Faculty of the Golisano College of Computing and Information Sciences in partial fulfillment of the requirement for the degree of Master of Science in Computer Science.

Approved by:

Advisor: Dr. Roger Gaborski — Roger Gaboski 4/19/2004

Reader: Dr. Michael Van Wie — Michael Van Wie 4/19/04

Observer: Dr. Edith Hemaspaandra — Edith Hemaspaandra 4/19/04

April 2004

Thesis/Dissertation Author Permission Statement

Title of thesis or dissertation: Development of a Human Visual System with the Ability to Detect Inconsistent Events

Name of author: Jeremiah D. Brazeau
Degree: Master of Science
Program: Computer Science
College: Golisano College of Computing and Information Sciences

I understand that I must submit a print copy of my thesis or dissertation to the RIT Archives, per current RIT guidelines for the completion of my degree. I hereby grant to the Rochester Institute of Technology and its agents the non-exclusive license to archive and make accessible my thesis or dissertation in whole or in part in all forms of media in perpetuity. I retain all other ownership rights to the copyright of the thesis or dissertation. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

Print Reproduction Permission Granted:

I, Jeremiah D. Brazeau, hereby **grant permission** to the Rochester Institute Technology to reproduce my print thesis or dissertation in whole or in part. Any reproduction will not be for commercial use or profit.

Signature of Author: Jeremiah D. Brazeau Date: 4/19/04

Print Reproduction Permission Denied:

I, _____, hereby **deny permission** to the RIT Library of the Rochester Institute of Technology to reproduce my print thesis or dissertation in whole or in part.

Signature of Author: _____ Date: _____

Abstract

Current research has shown that it is possible to create a general purpose learning system that models the first few layers of the human visual system. The goal of this thesis is to build upon that idea and produce a computational model capable of learning different aspects of visual information. Simulations show that the system is capable of learning and distinguishing different types of motion. Results also give one explanation, consistent with current experiments, explaining how the human visual system learns information and as such may produce the capability to predict behavior in future experiments.

Acknowledgements

I would like to take the opportunity to thank the people who have supported me through my Masters experience.

Contents

List of Figures

List of Tables

Missing Page

Chapter 1

Introduction

Although much is known about the human visual system, few models have been produced that are capable of processing information under circumstances as diverse as those encountered by the human eye. One problem that computers face is that they simply are not yet capable of processing so much information so quickly, perhaps because the human brain is more complex, having an estimated 10^{15} synapses compared to the Pentium 4's 42 million transistors (Kandel *et al.*, 1991; Bednar, 2002).

Although computers are not as complex, many models have been proposed that are capable of learning different aspects of the human visual system. Some of these models look for predefined patterns that occur within images (for example, edges are often found using a difference of gaussians approach). Although these approaches are good at finding specific patterns within images, they are unable to adapt to or to find novel patterns. Other models try to make use of neural networks to look for more general patterns within image data. These approaches generally have a better chance of finding patterns that were not predetermined; however, they may require training before they

begin to give meaningful results.

From a biological point of view, the approach of neural networks allows for the ability to adapt to different environments. One question still remains: how is the human visual system able to learn so quickly, given that it has never been presented any visual stimulus? Recent experiments show that the visual sensory system in mammals has gotten around this problem by specifying, in the genome, a method of training (Bednar, 2002). The advantage of specifying a form of training instead of actual capabilities seems to be that it allows for the system to see many features early on while also allowing for a system that is able to adapt to a number of different environments. Such training occurs before birth and can prepare the visual system for what it will encounter later in life. Given that training occurs within the visual system before it is introduced to an open environment, some level of proficiency in sight can be gained before birth. Research also suggests that in many mammals a special directionally selective set of cells exist. These cells, combined with learning that occurs prenatally, could lay the foundation for all that humans see in terms of motion.

Experiments have shown that when information into the visual system is disrupted, the resulting system is not capable of perceiving all forms of visual stimulus. Of particular interest is the ability to use this fact to detect novel events that occur within a series of images. As simulations will show, it is possible to train a system in one environment with the intention of later placing the system in a similar environment, such that the amount of activity that occurs within different parts of the simulated visual system can be used to determine the novelty of the new environment.

This thesis looks at one model of the visual system, which accounts for

such prenatal activity, and furthermore is capable of becoming orientation selective. An attempt is made to expand that model, proposing a system that is capable of distinguishing different types of directional motion. Finally, the amount of activity in the system is investigated to see whether or not it is possible to determine the occurrence of novelty within images.

1.1 Approach

A system that models the different areas of the human visual system including the retina, LGN, V1 and MT areas is constructed and compared to what is already known about the human visual system, with a hope that such a system will be able to make predictions about the outcomes of future experiments. The visual system was chosen because it is a well studied sensory system, and because many models of visual detection currently exist.

The system will be based on work published by James Bednar, which describes a system named HLISSOM (Bednar, 2002). This system was created to study the pattern generation hypothesis, which states that learning occurs prenatally in such a way that the visual system comes prewired, and yet contains the ability to adapt to the environment in which it is later placed. HLISSOM models only parts of the retina, LGN and V1 layers of the visual system, and only looks at orientation processing (Bednar, 2002). The system created for this thesis expands upon this model in order to process motion. In particular, this thesis looks at directional processing since it is well studied in a number of different species.

The simulations focus on the ability of the system to learn different aspects of its environment and use that knowledge to guide future decisions. Of

particular interest is the system's ability to distinguish between what it has seen in the past and what it has not yet seen. This will be the main method of testing the systems success and/or failure. Further comparisons will be made to visual detection systems that do not rely on learning.

Chapter 2

Background

Two topics are most relevant to this work. The first is a strong understanding of the human visual system. This includes understanding the different areas of the human visual system, how they are connected and what each of the areas function is (Blake *et al.*, 2003; Kandel *et al.*, 1991; Rosenzweig *et al.*, 1999). The second is Self Organization. Self Organization is a machine learning technique that requires no user input in order for the system to learn useful information. The main form of self organization that will be used in this system is modified Kahonen Maps and Hebbian Learning (Kohonen, 1997; Hebb, 1949).

2.1 Processing Visual Information

The human visual system consists of a number of independent layers that communicate with one another. Each of these layers is thought to be made up of cells specific to the functioning of that layer. Each layer is also connected to a number of other layers providing for information to be combined and allowing for a system of feedback.

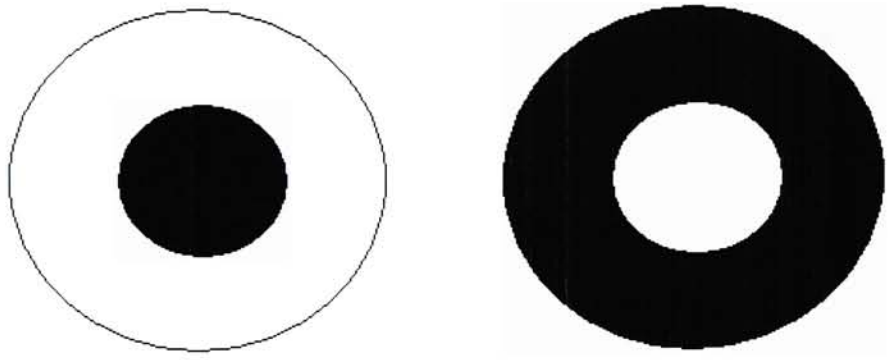


Figure 2.1: The above images are examples of ON-center and OFF-center cells which are types of retinal ganglion cells. These cells exist between the back of the retina and the lateral geniculate nucleus (LGN).

In order to simplify the model, only monochrome information is kept. As such, this section focuses on parts of the visual system that make use of monochrome information and ignore the parts of the visual system required for color processing.

2.1.1 The Layout of the Human Visual System

The first layer in the visual system is the retina. The retina is responsible for taking light, varying by wavelength and intensity, and producing an internal response. The retina is composed of 2 different types of cells that respond to different light information: the rods and the cones. The rods are responsible for seeing light whose wavelength is around 500nm. The rods are much more sensitive to intensity and as such allow us to see even when little light is available. The cones come in three types: those that peak around 445 (blue),

those that peak around 540 (green) and those that peak around 570 (red). Activity that these cells detect is encoded, and patterns of electrical activity are generated in neurons called the retinal ganglion cells (Rosenzweig *et al.*, 1999).

Retinal ganglion cells respond most strongly to edges between light and dark areas and as such perform a type of edge detection. Two specific retinal ganglion cells that are known to exist are ON-center and OFF-center cells shown in Figure 2.1. ON-center retinal ganglion cells respond most strongly when there is a light area surrounded by a darker region. This entire region is known as the retinal ganglion cell's receptive field (RF). The OFF-center retinal ganglion cells by contrast have the strongest response when there is a dark spot that is surrounded by a light region.

As information passes through the retina it then moves up the optic nerve to the lateral geniculate nucleus (LGN), located in the thalamus. As with the retinal ganglion cells of the retina, there are two specific types of response cells in the LGN (ON and OFF). The ON-center retinal ganglion cells connect to the ON cells of the LGN and OFF-center retinal ganglion cells connect with the OFF cells of the LGN. Each of the cells in the LGN is connected to a region of cells, its receptive field, such that when a response is generated in that region, the specific LGN cell will also respond.

From the LGN, information travels to the primary visual cortex (V1). The V1 layer is the first layer that is orientation selective. Just as the LGN layer is connected to the retina, the V1 layer is connected to the LGN layer such that each cell of the V1 layer is connected to a region within the LGN. The types of cells in V1 differ from those found in previous layers. There are two types of cells commonly found in V1: simple and complex. Simple cells can

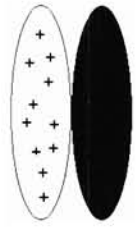
be thought of as edge and bar detectors. Figure 2.2 shows 4 different simple cells. The first two are selective of edges while the second two are selective of bars, or lines, that run through their receptive field. Complex cells still respond to orientations; however, these cells may only respond when an edge exists moving in a particular direction. This said, V1 is also one of the layers in the human visual system that is sensitive to direction. Cells, such as some complex cells, that are selective of direction are generally termed DS cells (Directionally Selective) (Blake *et al.*, 2003).

From the primary visual cortex, information branches to a number of other areas along two main pathways. This first pathway leads to the temporal cortex. The temporal cortex is responsible for our understanding and recognition of objects. The second pathway leads to the parietal cortex. The parietal cortex is responsible for processing and understanding motion.

The first stop in the parietal cortex is MT (Rosenzweig *et al.*, 1999). MT is most responsive to edges that move in a specific direction. The receptive fields in MT tend to be much larger than those in V1 (in fact they can be up to 10 times as large). In order to account for such selectivity, MT must be given information from more then one time period. Such information is provided through feedback loops and delayed responses.

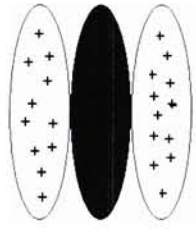
Once through MT, information moves to a number of other regions. At each step the information is reprocessed in a more refined manner than in the area that proceeds it. In this way, more complex patterns of motion can be observed as one proceeds farther along the motion pathway (Grzywacz & Merwine, 2002).

At this point the visual system can be laid out as shown in 6.2.

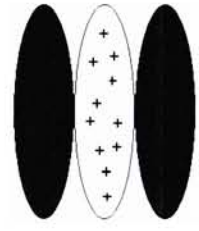


(a) Vertical Edge Detector

(b) Horizontal Edge Detector



(c) Dark Bar Detector



(d) Light Bar Detector

Figure 2.2: (a) and (b) are examples of neurons in V1 that act as edge detectors. (c) and (d) are examples of neurons in V1 what will respond most strongly to vertical dark and light bars. The pictures above describe the receptive fields of these neurons. In other words a neuron in V1 will respond when the particular pattern exists within its receptive field in the previous layer (in this case the LGN).

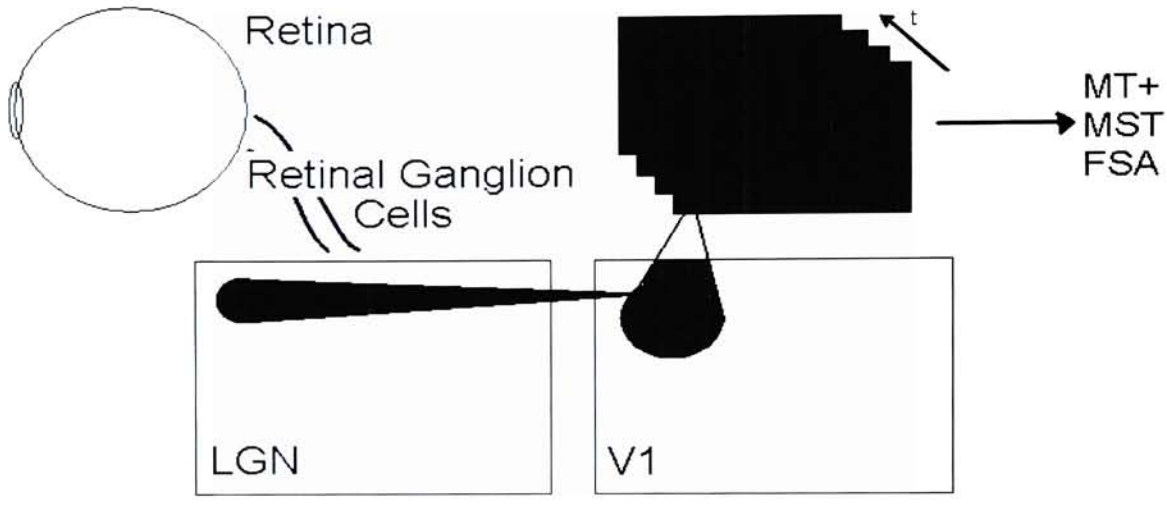


Figure 2.3: This figure represents the layout as discussed so far. Information travels in through the eye to the retina. From there information moves down the retinal ganglion cells and onto the LGN. From the LGN information continues on to V1. V1 is the first area to show orientation selectivity which is possible because of the structure of its receptive fields. There is also evidence that V1 is directionally selective in humans, as it is in other species. From V1 information moves to the MT region where motion processing is started. From here information travels to a number of regions including the MST, FSA and MT+ subregions.

2.1.2 Motion Processing

Newton once said that in order to understand motion one needs to know direction, speed and acceleration. As humans we are able to distinguish motion. We are also capable of differentiating speeds of objects. When it comes to acceleration however, we are not as capable.

Newton was describing motion in a scientific sense; however, his ideas also relate to a human's perception of motion. A number of different things need to happen in order for the brain to get a picture of motion. Seeing motion is not as simple as noticing an object move across a screen. For example, consider what happens if you watch a car driving down the street as you move your head. You have a sense that the car is still moving, even if it remains within your sight. Another example would be if you were sitting in the car. Even if you pass someone who is standing, you are able to tell that they are not moving, even though they move across your visual field. In order to accomplish all of this a number of areas of the brain play a role. Two of the major players are the MT and MST areas.

William T. Newsome performed experiments on monkeys to help show the function of MT (Newsome & Pare, 1998). Prior to the experiment, monkeys were trained to report their perception of direction while watching dots move in specific directions. During the experiment the same patterns of dots moving were shown to the monkeys and brain activity was recorded. The results showed a direct correlation with the activity with the cells in MT. Further experiments showed that a monkey's responses could be altered by injecting MT cells with low levels of current (Grzywacz & Merwine, 2002).

The MT area is one of the first areas that shows preference for motion in

one direction or another independent of the orientation of the object that is moving. Such a result may indicate that the MT uses information from the output of V1, and other areas, from different times. In this way it would be possible for the directionally selective cells in MT to become active when objects form the appropriate orientation through time.

The MST region of the brain is also very important in our understanding of motion. This area seems to be responsible for understanding motion by using information that is not visually available. It is this area that helps us to understand that when we turn our head everything in a room remains still even though it is crossing our visual field. In terms of understanding motion, this is vital.

2.2 Models of Self Learning

A number of different models have been produced for learning information about arbitrary systems. Many of these models require some form of interaction to take place. For the visual system this is not desirable since we as humans do not require any interaction with our environment to initially see. Nor do outside forces tell us whether or not what we are seeing is correct or incorrect. Therefore, this other forms of computer learning must be investigated. Two such systems are Self Organizing Maps and Hebbian Learning. Both are capable of recognizing patterns that exist when given input, and both attempt to form different categories based on these patterns, such that when new input is given to the system, it can be placed in the appropriate category.

2.2.1 Self Organizing Maps

The previous sections have briefly outlined the layout of different components of the human visual system. Each of these different areas plays an important role when looking at developing a model that is capable of processing visual information. What still needs to be explained is how all of these different areas will contribute to learning in a computer simulated model. It is clear that as humans we do not have to be told how to see; rather, over time we simply learn that ability on our own. In fact we are able to recognize some things when we are first born. In order to accomplish this within a computational environment, some form of self organization must be present.

Self Organizing Maps (SOM), also known as Kohonen maps, were first developed by Teuvo Kohonen around 1989. Kohonen maps are capable, without any previous knowledge, of associating concepts with categories. In particular, each concept becomes associated, over time, with a location on the map. Concepts that are similar are located near each other on the map, while dissimilar concepts are further apart on the map.

The obvious question that remains is how does the map update itself in such a way that no interaction is required? This has to do with the setup of the map. You can think of the map as a 2 dimensional matrix. In this matrix each cell has a vector of weights. There is one weight for each of the attributes associated with an input vector. The weights of each cell are initially random.

Like most neural networks, a Kohonen map learns over a series of rounds. During each round all input vectors are read in some arbitrary order. Once read, an input vector is processed by first selecting a winner from the Kohonen

map and then updating the surrounding cells of the winner. In order to pick the winner, the input vector is compared with all weight vectors in the map. The winner is that vector in the map which is most like the input. To determine how close two vectors are a distance measure is defined. Some well-known measures of distance are the Euclidean distance and Manhattan distance. During the update phase of this process, each of the vectors that surround the winner is updated to more closely resemble the winner. The size of this surrounding area, and the amount by which these surround vectors, are changed is slowly reduced during each round such that well-defined groups can be formed.

Self Organizing Maps can be described by the following algorithm:

1. Read an input vector.

2. Determine which of the cells in the Kohonen map is the winner.

   $\|input - node_{winner}\| = \min_i \|input - node_i\|$

   Here distance for a single node can be defined as:

   $d_{out} = \sum_{in}(input_{in} - node_{in})^2$

3. Modify the weights w_i of the winning node and its neighbors so that they more closely resemble the input.

   Input vector = p_i

   $h(\mathrm{r},\mathrm{t}) = a(\mathrm{t})v(\mathrm{r})$

   $a(\mathrm{t})$ is the learning coefficient

   $v(\mathrm{t})$ is the neighborhood function

   $w_i(\mathrm{t}+1) = w(\mathrm{t}) + h(\mathrm{r},\mathrm{t})(p_i - w_i(\mathrm{t}))$ if $i \in$ neighborhood

$w_I(\mathrm{t}+1) = w(\mathrm{t})$ if $i \notin$ neighborhood

4. Decrease the size of the neighborhood of the winning nodes
5. Decrease the learning coefficient, *a(t)*, which controls the amount of change of each of the neighbors in the winning neighborhood.
6. Learning finishes when the learning coefficient is zero.

2.2.2 Hebbian Learning

Hebbian Learning is based on the principles of a work published by the Canadian neuropsychologist Donald O. Hebb. Hebb states:

> When an axon of cell A is near enough to a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased (Hebb, 1949).

In other words, Hebb felt that if receptors are subject to repeated stimulation, they will learn to act together as a closed system, even after that stimulation is no longer present.

This idea of learning has carried over into computer science and is termed Hebbian Learning. Hebbian Learning is described as an unsupervised learning algorithm in which the weight between two neurons will increase if both the source and target neurons are active at the same time (Medler, 1998). A simple formula for this would be:

$$\Delta W_{ij}(t) = \gamma * x_i * x_j$$

where W_ij is the weight from neuron x_i to neuron x_j. A more general model might also take into consideration time and learning thresholds.

2.3 Early Development of the Visual System

Although the structure of the visual system is well studied, there is still much debate over how this structure is defined within the genome. There are two major encampments, those that feel that genetics act as a blueprint for the structure of the visual system and those that feel environmental influences are responsible for its structure. These two sides are found throughout psychology and has become known as the Nature vs. Nurture debate.

As most occurrences of this debate, there is no clear evidence for either side; rather, it is more likely that some of the human visual system follows a blueprint that is predefined in our genetic code while other parts of our visual system adapt and are formed based on environmental influences. The following sections describe how both genetics and the environment are able to work together to form a system capable of perceiving visual information.

2.3.1 Genetic Influences

In building a strong case for the role genetic influences play on the production of a visual system, it can be useful to acknowledge trends that occur within cultures or families. For example, if twins are separated at birth, what are the chances that both are near sighted, or both are color blind? These questions have long been studied by psychologists as a way to determine the significance of genetic factors in making us who we are. As a result there are a number of important experimental results that show the importance of genetic factors in defining the visual system. For example, studies have been able to show that myopia (near-sightedness), hyperopia (far-sightedness) and color blindness all are linked to genetic influences (Coon, 2000).

Physical structure is also clearly defined, and it is known that the visual system amongst different species is not the same. For example, turtles have far more directionally selective cells then humans. Such distinctions exist long before birth and thus before any visual environmental influences are present. There is very little difference in structure within a single species.

Receptivity to orientation and motion may also in part be defined genetically. For example, humans are not very good at learning to correctly perceive objects if there is a time delay. If you watch a poorly made movie you may notice that there is a delay between the sound you hear and the visual movement of the mouth. No matter how long you watch the show, you will never be able to correctly perceive the movie.

In terms of orientations, a few experiments have been able to show the role that genetics plays in the development of the visual system. Advances in medical imaging have allowed for full orientation preferences to be measured in animals. Such studies have been able to show that orientation preferences exist before there is any visual experience (and thus any environmental influences). Furthermore, these studies have shown that such orientation preferences closely mirror those found in adults (Crair & Stryker, 1998).

2.3.2 Environmental Influences

Chalupa and Rhoades found that visual deprivation significantly changed directional selectivity, speed preference, and receptor field organization. When they looked at hamsters raised in stroboscopic illumination there were signs of visual deprivation, showing that an environment can influence the visual system (Chalupa & Rhoades, 1988).

Perhaps more well known is the work of Nobel Prize winner Torsten Weisel. In one of his lectures he describes experiments that he performed with David Hubel while investigating the response properties of cortical cells in cats during postnatal development. The experiments, performed on kittens, required that the eyelids of a subject be fused shut. He found that if only one eye was kept shut during the first three months after birth, and then allowed to open, the kitten was blind in the eye that was initially closed. If the same experiment was performed on an adult cat, blindness did not occur. Such work provides evidence that environmental influences are important for proper development of the visual system (Wiesel, 1982).

More recent experiments on kittens raised without sight in one eye have shown the development of non-specific lateral interactions for that eye. If, on the other hand, strabismus, more commonly known as crossed-eyes (a condition in which a both eyes can not be aligned simultaneously), was present in kittens, then the lateral connections were found to become patchier (Kasamatsu *et al.*, 1998; Gilbert *et al.*, 1990; Lowel & Singer, 1992).

Blakemore and Cooper were able to show that kittens raised in environments with only vertical stripes developed in such a way that they were good at seeing vertical stripes but bad at seeing horizontal stripes. In other words it is possible that edge detectors in V1 only developed for vertical lines (Blakemore & Cooper, 1970).

Finally, Stratton was able to show that the entire visual system can be affected by environmental influences. In his experiments, goggles were used to invert an incoming image. The question was whether or not an observer would be able to perceive visual information appropriately, and the answer is yes. After several days of wearing the goggles, a subject perceived his/her

environment in the same way he/she had before the goggles were introduced (Stratton, 1897).

2.3.3 Activity Before Sight

It is important to understand the connection that genetics plays in the development of visual perception both through genetically defined structures and genetically defined activity. Up to this point we have seen that the structure of the visual system can be defined by genetics, and that the environment can have long lasting effects on visual perception; still, newborn babies are capable of perceiving some objects as soon as they open their eyes. The question then is how perception is defined genetically before birth.

Experiments have shown that spontaneous neural activity occurs in many different cortical and subcortical areas. Recent experiments have shown that such activity, if disrupted, can have long lasting developmental effects.

One example of spontaneous neural activity is retinal waves. A number of experiments have been able to link retinal waves with the development of the visual system. One such experiment disrupted all retinal waves. As a result the LGN failed to develop as it should, and neurons in V1 were found to be less selective of orientation (Chapman, 2000; Chapman & Stryker, 1993). Other recent experiments have been able to show that retinal waves can also be instructive. For example, one experiment involved artificially activating a large number of axons in the optic nerve of ferrets, so that activity in these areas still existed although the activity was not the same as naturally occurring retinal waves. The result was that neurons in V1 were less selective of orientation, which shows that it is not only activity that is important, but

also the type of activity. (Katz & Shatz, 1996).

Another example of spontaneous neural activity is ponto-geniculo-occipital (PGO) waves that occur during rapid-eye-movement (REM) sleep. These waves travel from the brainstem to the LGN, visual cortex, and a number of subcortical areas. In human adults, these waves are strongly correlated with dreams and their associated imagery (Marks *et al.*, 1995). Experiments on kittens have shown that when visual stimulation is only allowed in one eye, by fusing the other eye shut, the LGN and V1 are affected (as described previously). When combined with the disruption of REM sleep the effects on the LGN and V1 areas have been found to be stronger indicating that PGO waves may help visual development even when visual impairments exist for a short while (Marks *et al.*, 1995).

Bednar has spent a great deal of time looking into the effects of spontaneous neural activity within the visual system. His work stems from research showing that such activity occurs within many developing cortical and subcortical areas, including the visual cortex and retina. His development of a computer simulated visual system has been able to further show that such stimulation, when it occurs prenatally, can ready a system for interacting with its environment (Bednar, 2002).

2.4 Conclusion

This chapter has looked at the layout of the visual system and at a number of computer algorithms that are useful in modeling such a system including Self Organizing Maps and Hebbian Learning. Finally, we discussed how the human visual system is thought to develop, including conflicting views

regarding the roles that genetics and the environment play.

Chapter 3

Discussion of Related Work

The focus of this thesis is the development of a computational model for the perception of orientation and motion. To date a large number of models have been produced, some more biologically oriented than others. Most of these systems store information in maps, which support the belief that many of the lower-level areas in the human visual system are organized retinotopically. That is, each of these areas is stored as a map which is capable of preserving the topography of the retina. In this section a number of the models that currently exist are presented. How well they model the different aspects of the human visual system, described in the previous chapter, is also discussed.

3.1 Computation Models for Learning Orientation

A large number of computational models of the human visual system exist, and as computers have been capable of performing calculations more quickly these models have evolved to take advantage of the increase in computational power. What follows is a discussion of these systems, their abilities in generating orientation maps, and how closely each system models a biological

representation of the human visual system.

3.1.1 von Malsburg's Method: Hebbian Learning

In 1973 von Malsburg was able to show that columns of orientation-selective neurons could develop, without supervision, from oriented binary bitmap patterns. Like most models that follow, von Malsburg's model was based on Hebbian Learning.

This model formed a strong basis for most of the following models and contained many of the characteristics found in later models, including the acknowledgement of lateral connections (in this case they are fixed strength), the treatment of the retina and cortical areas as two-dimensional arrays, and a method of assigning a value between connections which describes that connections strength (Bednar, 2002; von der Malsburg, 1973; Rochester *et al.*, 1956).

Like many of the models based on von Malsburg's, most of the problems stem from the fact that lateral connections are of fixed strength. One the other hand, this particular model is capable of creating a map whereby neighboring cells share orientation preference, and as described by Bednar has the capability to produce pinwheels within the orientation maps, which had not previously been accomplished (Bednar, 2002).

3.1.2 Linsker's Method: Information Maximization

Linsker created a model based on local Hebb-like synaptic modification rules. The modified rule used is

$$\delta w_j = k(yx_j + bx_j + cy + d)$$

where yx_j is the Hebbian term and k is the learning rate. Each of the three remaining terms modify the network to follow some set of constraints where b, c and d are different constants (Linsker, 1986; Rolls & Deco, 2002).

The network is put together as a multi-layer system as shown in Figure 3.1. Each layer, except for the first, looks at an area of the previous layer forming Receptive Fields. The first layer acts as input similar to the cones and rods in the retina of the eye.

In order to train the network, random firing takes place. Over time each of the layers starts to form preferences. Rolls and Deco have observed that by the third layer, the formation of receptive fields that resemble center-surround cells could be found, and by the sixth layer, the receptive fields had become elongated and responded best to oriented edges or bars.

Although the system was capable of developing receptive fields that were orientation selective, the overall map was disorganized. This was due to the lack of lateral connections. As a result cells in the sixth layer did not share similar orientation preferences when neighboring each other. This leads to redundancy within each layer. Another problem also exists because of the way in which the system was randomly trained. When the system is first placed in a real environment there is no guarantee that it will be capable of perceiving anything in this new environment (Rolls & Deco, 2002).

A more fundamental problem with this system is that it does not address higher order features such as a correlation between multiple edges. As such, this system is only capable of learning simple orientations.

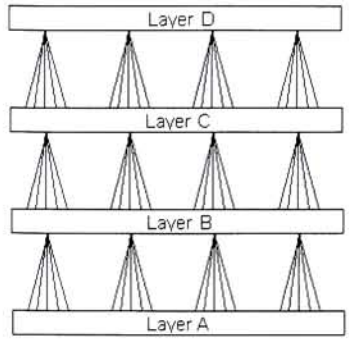


Figure 3.1: Linsker organized his system as a number of layers. The connections between layers are linear so it would be possible to perform all steps at once. Only 4 layers are shown here but more are possible. By the sixth layer the responses resemble those found in V1's simple cells although neighbors do not show similar orientation preferences.

3.1.3 SOM Models

Neither of the previous two models were capable of storing data retinotopically. This is in part because the two previous models generally modeled only a small number of neurons. Specifically, because of computational constraints, the von Malsburg model was only capable of expressing a few neurons, all with overlapping receptive fields. By the late 1980's and early 1990's, the amount of computational power available had grown immensely. As a result, new models were produced, and many of these made use of a more abstract network known as the Self Organizing Map (SOM) produced by Kohonen.

Work by Durbin and Mitchison (along with others) used SOM's and modified them to work within the orientation domain. The results were promising, capable of producing orientation maps that were comparable to those found in the visual system and included properties such as pinwheels, fractures and linear zones (Bednar, 2002).

3.1.4 Correlation-based learning models

Correlation-based learning stems from the observation that the visual system may be linear. If this observation were true, then a function could replace sequences of input patterns. Such a system would greatly improve the speed of calculations when compared to the two previous models, which involve incremental Hebbian learning. This approach was first suggested by Miller in 1994.

This model made a number of improvements in terms of what parts of the visual system were modeled. For one, on and off center cells in the LGN were modeled, and as a result multi-lobed receptive fields formed (no previous model accomplished this). Although a step in the right direction, the maps produced by this model were not as realistic. The receptive fields were close to binary whereas receptive fields observed in animals have smooth edge transitions. As a result the responses achieved by the more computationally intensive Hebbian learning method proved more realistic and biologically plausible (Bednar, 2002; Piepenbrock *et al.*, 1996).

3.1.5 RF-LISSOM

SOM models, although more computationally expensive, have produced more realistic results. One thing that these models have lacked in is an explanation of lateral connections. SOM models use a fixed neighborhood, but some suspect that lateral connections formed prenatally help form orientation maps (Bednar, 2002). As a result a new model is needed that accounts for the patchy lateral connections that are formed prenatally.

RF-LISSOM (the Receptive-Field Laterally Interconnected Synergetically

Self-Organizing Map) was the first model to accomplish this (Sirosh, 1995). This model was based on previous SOM models but no longer approximated lateral connections via a fixed neighborhood. Instead, incremental Hebbian learning was used to develop both lateral and afferent connections. As a result, redundancy within maps, produced by previous models, was reduced.

The major drawback to this system is that responses on real images are less realistic, since it does not account for both ON and OFF cell layers. Bednar has shown that this limitation must be overcome when working with natural images (Bednar, 2002). As a result the system has been modified and is discussed in the next section.

3.1.6 HLISSOM

HLISSOM was produced as an extension to the LISSOM model. It includes not only the ability to learn lateral connections, but also the ON and OFF center channels first introduced in the Correlation-based learning models. The resulting model has been shown to work with real images, an ability that previous models have lacked. It is also capable of sending information to higher cortical areas. In particular, it models face-selectivity (Bednar, 2002).

3.1.7 Conclusion

Although this and all previously discussed models are capable of modeling orientation selectivity, none of these models extend to the perception of motion. In particular, although orientation selectivity exists within V1, direction selectivity also exists and therefore should be accounted for in these models.

The next chapter is based on the HLISSOM model and looks at expanding that model to account for directional selectivity in the V1 and MT areas.

3.2 Computation Models for Learning Motion

A number of people have also investigated motion perception. Many of these models are based on simplifications of the human visual system and do not as closely follow a biological approach. Discussed below are a number of these systems.

3.2.1 Complete Motion Processing Models

A few models have been developed that encompass the entire motion processing pathway of the visual system. These models have shown to be capable of responding to motion in much the way we as humans respond to motion. Many of these systems directly specify the layout of the visual system. As a result, these systems do not internally learn responses, and are only an estimation of what occurs naturally.

Experiments on these systems have shown that it is possible to produce a system that can determine not only simple but complex motions such as rotations, contractions and expansions. These systems can also determine simple motion such as speed and direction (Pomplum *et al.*, 2002; Grossberg *et al.*, 2001). As a result these systems are useful in explaining human behavior in terms of motion as a whole, and some interesting results in terms of attention have been found (Pomplum *et al.*, 2002).

What these systems lack is the ability to learn motion. They are preprogrammed and therefore are do not help to explain how motion perception is

learned. For example, if this system were only told about horizontal motion, it would never be capable or learning vertical motion.

3.2.2 One Extention of HLISSOM

Bednar has discussed a possible expansion to the HLISSOM model which would provide it with some sense of direction. The extension involves spatio-temporal effects at the LGN and V1 layers. As proposed, the model could be expanded to include 4 separate LGN layers each responding to a different point in time.

This model has shown the formation of long range lateral connections between neurons that share similar direction preference. Clearly this is a step in the right direction, but the model only goes as far as explaining interactions between neurons at the V1 layer. As such, more work is needed to expand this model to account for directional and speed selectivity at the MT layer.

It is not clear at this time whether or not this model is capable of working on real images, as no data has yet been published using real images as input (Bednar & Miikkulainen, 2003).

3.2.3 Layer Specific Models

Although looking at the entire visual system is ideal in understanding its overall function, some research has focused on just one layer of motion processing. These models are ideal when wanting to gain insight into the functioning of a specific layer. For example, researchers at Boston University have developed a model capable of describing optic flow selectivity in the MST layer of the

visual system. Their model uses a back-propagation network to investigate the different interactions that take place in MST, specifically looking at the interaction between locally complex motions with global motion. A major drawback is that the system requires supervised learning (Beardsley & Vaina, 1998). Similar models have also been produced for other specific areas of the visual system (Simoncelli & Heeger, 1998).

3.3 Conclusion

Previous models have shown the ability to model the beginning stages of the visual cortex in a generic way. Other models have shown that it is also possible to model higher cortical areas of the visual system using linear models. What these models have not yet shown is the ability to generalize past the V1 layer in order to process motion. Nor have they been used to aid in novelty detection within image sequences. The next chapter will discuss a model that will attempt to overcome these shortcomings and show that Hebbian learning can account for both orientation and direction selectivity within the visual system.

Chapter 4

A Model of Motion Processing: DS-LISSOM

This thesis presents a modified version of HLISSOM (Bednar, 2002). HLISSOM is a biologically inspired model of the human visual system focusing on the information pathway from the retina to V1. The new model presented here attempts to expand HLISSOM to model directional selectivity as it is found at the MT layer. The expanded model also attempts to detect novelty by looking at the amount the system must change in order to incorporate the new information. This chapter focuses on presenting the structure of the model, and discusses how each step is biologically inspired.

HLISSOM was chosen as the model to follow because of its strict adherence to emulating biological processes and its ability to work with real images. Also, few parts of the HLISSOM system are specified beforehand, and therefore it is possible to make connections between the way the model learns and experiments performed on animals. In later sections we will demonstrate this connection.

One major omission in this model is the lack of chromatic information.

Experiments have shown that the chromatic properties of an object have little influence on MT neurons' motion processing when there is sufficient luminance contrast (Thiele *et al.*, 1999). As such, in order to simplify the model, chromatic information is not processed.

4.1 Overview

In this section DS-LISSOM (Directionally Selective LISSOM) is laid out in its entirety. Later sections will focus on how each of the layers are developed. A pictorial representation of the layout of DS-LISSOM can be seen in Figure 4.1. Information is first processed by the retina. There are two such sources of information: real images which represent sight and synthetically generated images which represent spontaneous activity within the visual system (as discussed in previous chapters). Spontaneous activity will later be shown to train the system in a way that is biologically similar to the visual stimulation that occurs prenatally. From the retina, information travels toward the LGN. The LGN layer is designed to be most responsive to bright and dark spots within a given receptive field. To more accurately account for this information, the LGN layer is made up of two separate channels. One of the channels is most responsive to light spots, while the other is most responsive to dark spots. From the LGN, information will travel to V1. For this to occur, each neuron in V1 looks at a region within the LGN layer (its receptive field). As learning occurs, it will not only become sensitive to a particular orientation, but lateral connections will also be updated such that neighboring neurons will share similar orientation preferences. In this system, lateral connections to nearby neurons will be excitatory, while longer lateral

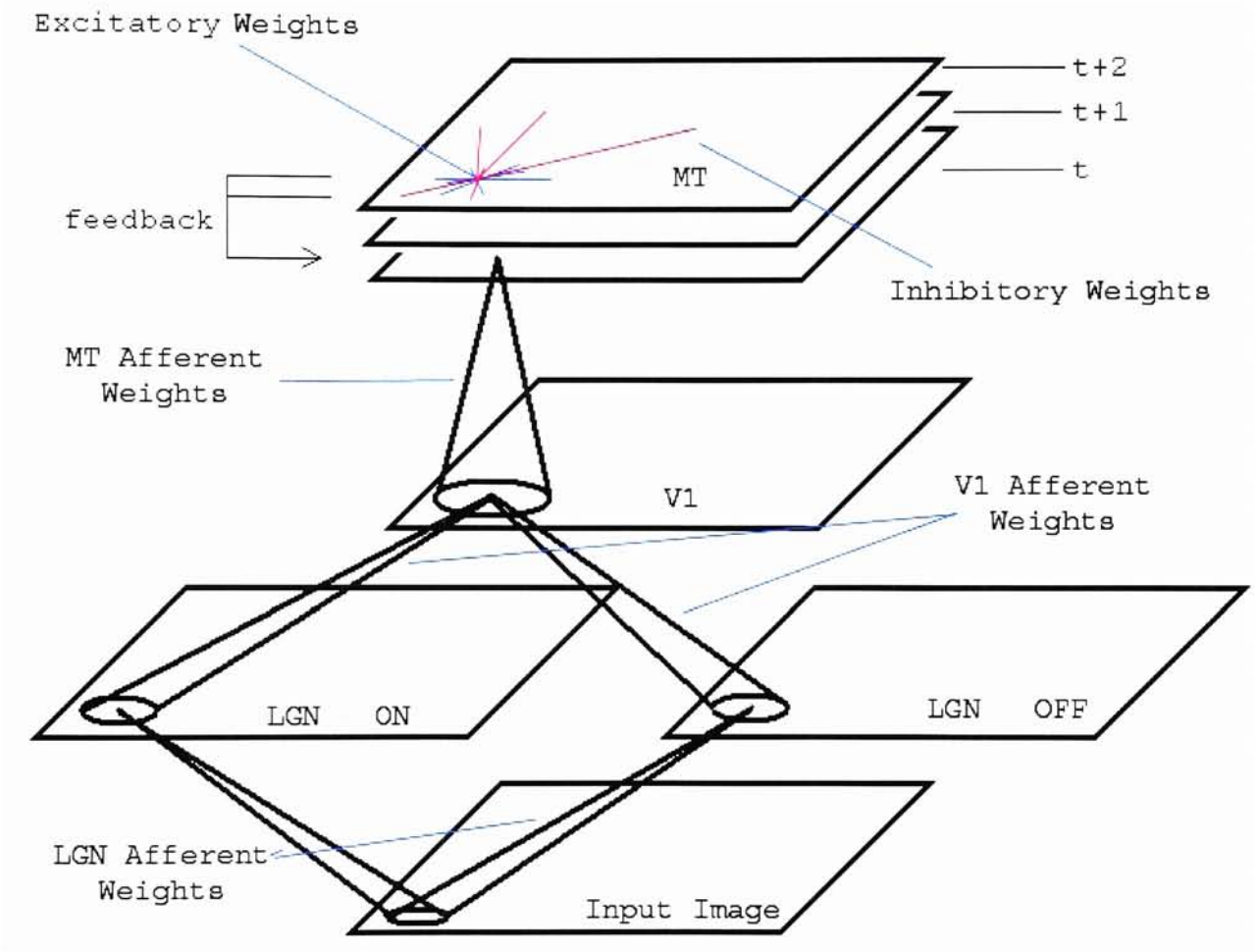


Figure 4.1: Each of the major areas of DS-LISSOM are labeled. This is a pictorial view of DS-LISSOM.

connections will be inhibitory. To move to the MT layer, a similar process takes place. Each neuron in MT pulls input from its receptive field in the previous layer. In order to account for directional selectivity within the V1 layer. an approximation is made by looking at the different responses V1 has at predetermined time steps. In this way, only edges which are thought to be moving are passed onto MT. Generally speaking, the receptive field of an MT neuron is larger then the receptive field of a V1 neuron. Lateral connections within the MT layer act similar to those in the V1 layer such that neurons with similar direction sensitivity tend to be close to one another. In order to account for directional information, MT must process information from a series of time steps. To accomplish this, MT feeds back information from the previous steps to itself.

The following sections will look at how each of the layers is constructed. In each section the following symbols are used

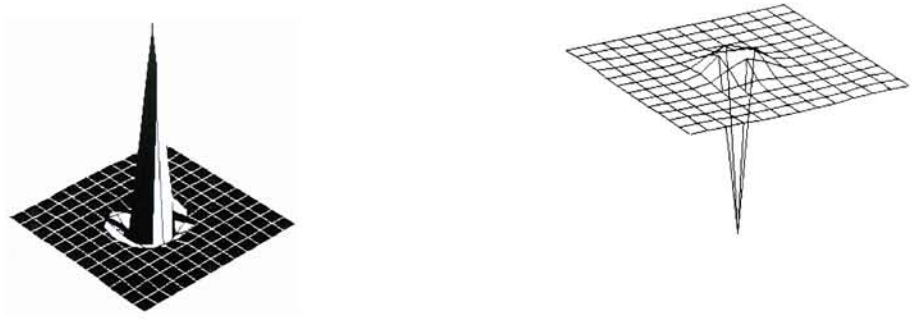
- η is the final activation of a particular layer.
- ζ is the afferent response of a neuron.
- ξ is the activation of a cell in the receptive field of another cell.

4.2 Modeling Retinal Activations

Information, or activation, in the retina comes from one of two places depending on whether or not the system is thought to be in a prenatal or postnatal stage. During the prenatal stage the retinal activation consists of patterns of oriented Gaussians as shown in Figure 4.2. These patterns can be generated by the following equation:

$$\xi_{xy} = \exp\left(-\frac{((x-x_0)cos(\theta)-(y-y_0)sin(\theta))^2}{a^2} - \frac{((x-x_0)sin(\theta)+(y-y_0)cos(\theta))^2}{b^2}\right) \tag{4.1}$$

where a^2 and b^2 specify the length along the major and minor axes, (x_0, y_0) is the center of the Gaussian, and θ is its orientation. The x and y coordinates are chosen randomly such that the Gaussian occurs within the retinal area of size RxR. θ is also chosen randomly such that $0° \leq \theta < 180°$. A random number of Gaussians occur within each iteration of training. In order that Gaussians do not overlap when multiple Gaussians are present their centers



(a) ON Center Surround Gaussian (b) OFF Center Surround Gaussian

Figure 4.2: These images are examples of the oriented Gaussians which are used as the afferent weights from the input image to the LGN. These act as simple edge detectors as it is thought the Retinal Ganglion cells do.

are specified to be at least d_r units apart. Later chapters will discuss the advantage of training the system on specific images in order to solicit significant responses when looking for novelty.

Once the system is considered to move into the postnatal stage (after approximatly 1,000 to 10,000 prenatal iterations for a general learning system) the retinal activation is a response to real input images. These images are converted to grayscale images, which account for intensity contrast information. Experiments have shown that it is the luminance contrast channel which, when present, plays a large role in the detection of motion (Thiele *et al.*, 1999).

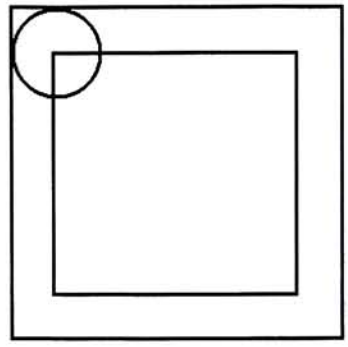


Figure 4.3: The circle represents the receptive field of a cell for example a cell in MT. This represents the fact that the preceeding layer, in this example V1, must be both longer and wider.

4.3 Processing from the Retina to LGN

Like its predecessors, DS-LISSOM is not interested in the learning processes that take place from the retina through the LGN. Therefore the connections to neurons in the ON and OFF channels of the LGN are set to fixed strengths which resemble those found in adult LGN cells. A Difference-of-Gaussians (DoG) model is used to accomplish this. An important design decision is to make sure that information is continuously stored retinotopically. Therefore the way in which each of the two LGN channels is organized is important. The (x, y) position of each neuron in the LGN layer corresponds with the center of each of the receptive fields. A consequence of this is that the LGN layer will be smaller then the retinal layer. The exact size of the LGN layer is the size of the retinal layer minus the diameter of an LGN neuron's receptive field as depicted in Figure 4.3. From the above information, the following equation can be derived for a neuron present in the ON channel of the LGN

$$\mu_{ab,xy} = \frac{\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_c^2})}{\sum_{xy}[\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_c^2})]} - \frac{\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_s^2})}{\sum_{xy}[\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_s^2})]} \tag{4.2}$$

In the above equation (a, b) is the position of the neuron, (x, y) is the position of each receptor within the retina with center (x_0, y_0), σ_c^2 is the width of the central Gaussian, and σ_s^2 is the width of the surround Gaussian. The receptor weights for the OFF channel can similarly be found by subtracting the center Gaussian from the surrounding Gaussian as follows:

$$\mu_{ab,xy} = \frac{\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_s^2})}{\sum_{xy}[\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_s^2})]} - \frac{\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_c^2})}{\sum_{xy}[\exp(-\frac{(x-x_0)^2+(y-y_0)^2}{\sigma_c^2})]} \tag{4.3}$$

A visual representation of the weights for these receptive fields can be seen in Figure 4.2. In order to prevent both ON and OFF cells from being active at the same location at the same time, each cell will be thresholded to have only positive activations. This is consistent with biological systems (Bednar, 2002).

The LGN is divided into two layers: one which represents the ON activation and another which represents the OFF activation from the retinal ganglion cells. Results given by Bednar have shown that when working with

real images this is important, since it will allow the LGN to respond to a wide range of brightness values (Bednar, 2002).

The activation of a single cell in the LGN layer can be calculated by the sum of activations in a cell's receptive field. From this the following is derived:

$$\eta_{ab} = \sigma\left(\gamma_A \sum_{\rho xy} \xi_{\rho xy} \mu_{ab,\rho xy}\right) \tag{4.4}$$

In the above equation, $\mu_{ab,\rho xy}$ is the afferent weight from a cell (x, y) to (a, b), γ_A is a constant scaling factor, $\xi_{\rho xy}$ is the activation of cell (x, y), ρ is the location of the receptive field, which can be either the generated input (PGO input) or image input from the photoreceptors, and σ is a piecewise linear approximation of the sigmoid activation function (allowing for faster calculations). The sigmoid activation function can be defined as follows:

$$\sigma(x) = \begin{cases} 0 & x \leq \delta \\ \frac{x-\delta}{\beta-\delta} & \delta < x < \beta \\ 1 & x \geq \beta \end{cases} \tag{4.5}$$

The activation function makes sure that there is at least a minimum amount of activity within the receptive field before the cell becomes active itself. This is extremely useful since many real images are noisy. The sigmoid function also ensures a maximum amount of activity in a particular cell.

4.4 Modeling V1

This is the first area of interest to the learning aspects of DS-LISSOM. It's implementation is identical to that implemented in the previously mentioned HLISSOM model with the exception that statistics are stored in order to determine whether or not a particular orientation has been seen before.

V1 cells are different than cells in the LGN in that their activation is a two-step process. The first step is similar to that of the LGN, and the activation of a cell can be specified by the following equation

$$\zeta_{ij} = \sigma\left(\frac{\gamma_A \sum_{\rho ab} \xi_{\rho ab}\mu_{ij,\rho ab}}{1 + \gamma_N \sum_{\rho ab} \xi_{\rho ab}}\right) \tag{4.6}$$

where ρ specifies the ON or OFF layer from the LGN, (a, b) is a cell within the receptive field of cell (i, j), $\mu_{ij,\rho ab}$ is the weight from cell (a, b) to (i, j) in the appropriate receptive field ρ, and $\xi_{\rho ab}$ is the activation of the cell (a, b) in the receptive field of cell (i, j) and on either the ON or OFF channel of the LGN specified by ρ.

The constant γ_N is used to make the activation of a cell more selective. This value often starts out near 0 and slowly increases over time such that cells become more selective. As a result of the value of γ_N it is important to keep all of the values spread out across the range from 0 to 1. For this reason the constant γ_A is introduced. Its purpose is to make sure that the afferent responses continuously fall between 0 and 1. If γ_A is not present while γ_N increases, then over time the activation of all cells will move toward

0, rendering the system inactive.

The sigma function is the same as that used when calculating activations in the LGN layer.

Once an initial activation response is calculated, the lateral excitatory and inhibitory weights get their chance to change the response. This is accomplished over a number of rounds using the following equations

$$\eta_{ij}(0) = \zeta_{ij} \tag{4.7}$$

$$\eta_{ij}(t+1) = \sigma\left(\zeta_{ij} + \gamma_E \sigma_{kl} E_{ij,kl} \eta_{kl}(t) - \gamma_I \sigma_{kl} I_{ij,kl} \eta_{kl}(t)\right) \tag{4.8}$$

The excitatory weights are defined as $E_{ij,kl}$, the weight from cell (k, l) in V1 to the cell (i, j) in V1 (and the inhibitory weights) are similarly defined as $I_{ij,kl}$. Each of the settling iterations helps to ensure that responses of nearby neurons will have similar responses, helping reduce the amount of redundancy in the map and aiding in the formation of orientation maps, as previously discussed.

4.5 Modeling MT

At this point things change, because temporal effects need to be considered. This is the first area in this model that is selective of direction. As such some form of feedback needs to be implemented to account for different periods in time. From a psychological point of view, this is thought to be accom-

plished in one of three ways: the Riechardt model, Barlow and Levick model, and the two asymmetrical pathways model. In the Riechardt model, past information acts to further strengthen current activations. The Barlow and Levick model proposes that instead of strengthening current activations, the interaction between past and current information is inhibitory. Finally, the two asymmetrical model combines both an idea of excitation and inhibition (Grzywacz & Merwine, 2002).

DS-LISSOM obtains temporal information using the Riechardt model, which states that the amount of activation at different periods of time have a multiplicative effect on the current activation of the neuron. In DS-LISSOM the amount of activation at a previous time step adds to the total amount of activation. The following equation models this behavior

$$\zeta_{ij} = \sigma\left(\frac{\gamma_A \sum_{\phi ab} \eta_{\phi ab}\mu_{ij,\phi ab}}{1 + \gamma_N \sum_{\phi ab} \eta_{\phi ab}}\right) \tag{4.9}$$

In the above equation, each ϕ is the time in which a cell from V1 (η_{ab}) is measured. For DS-LISSOM, four such layers of information are kept. Because of the changing weights within each layer, the actual activation is not stored; rather, it is recalculated using the current weights for that particular period of time. The following equations describe how each $\eta_{\phi ab}$ is determined:

$$\eta_{0ab} = ((\eta_{ab}(t) - \eta_{ab}(t-1)) > 0) * \eta_{ab} \tag{4.10}$$

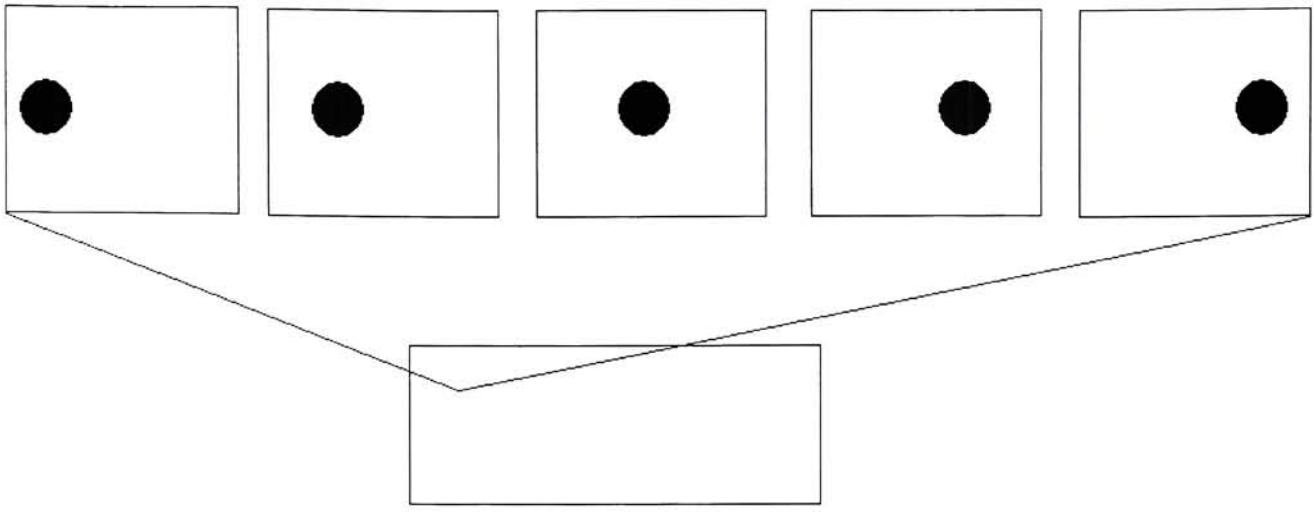


Figure 4.4: Each of the top images represent a receptive field of a single cell in MT. An MT's receptive field will draw data from multiple points in time. In the above example, an MT is looking at 5 different points in time. As an object moves the developing MT cell will adjust its weights such that it will respond most strongly when, in this example, a bright object moves to the right through its receptive field.

$$\eta_{\phi+1ab} = \eta_{\phi ab} \tag{4.11}$$

In this way ϕ describes how long ago a given activation occurred. Notice the way in which η_{0ab} is produced. The only activations that travel to MT are those activations which have changed from one time step to the next. This is meant as a simple way to model some of the directional selectivity that occurs in V1.

To think about this in a slightly different way, imagine that each of the layers is the actual activation at the current time step of the previous layer. The receptive field of a current cell is the combination of a receptive field in each of these layers, and a connection exists between the cell and each

of the cells in each of these receptive fields. The activation of the cell is the combined total of all of the activation in all receptive fields. Motion is learned because the system will learn where objects exist over time as shown in Figure 4.4.

Just like V1, the MT layer also goes through a number of settling iterations whereby the final response of a cell is formed according to the following equations

$$\eta_{ij}(0) = \zeta_{ij} \tag{4.12}$$

$$\eta_{ij}(t+1) = \sigma\left(\zeta_{ij} + \gamma_E \sigma_{kl} E_{ij,kl} \eta_{kl}(t) - \gamma_I \sigma_{kl} I_{ij,kl} \eta_{kl}(t)\right) \tag{4.13}$$

As in V1 the settling iterations will help reduce redundancy in the map produced by MT. This will also help ensure that nearby neurons have similar patterns of activity, which is a property of cortical responses.

4.6 Training the System

Up to this point it should be clear how cells become active within the system. What is still not clear is how the system is capable of learning in the first place. Learning occurs by recognizing the correlation between input patterns. In the above equations the μ, E and I responses are adjusted over time to adapt to these patterns. This occurs through a form of Hebbian learning using the following equation

$$w_{ij,mn}(t+\delta t) = \frac{w_{ij,mn}(t) + \alpha \eta_{ij} X_{mn}}{\sum_{mn}(w_{ij,mn}(t) + \alpha \eta_{ij} X_{mn})} \tag{4.14}$$

The goal of this equation is to increase the weight, $w_{ij,mn}$, if and only if the activation of a cell in the current level, η_{ij}, and the activation of another cell, X_{mn}, are high at the same time. For afferent connections X_{mn} is the amount of activation of cell mn in the previous layer while lateral connections look at the current level, η_{mn}. α is the learning rate, and it can be helpful to define different learning rates for each layer and for each type of connection. The normalization that takes place keeps connections from growing without bound.

The result of the learning procedure is lateral inhibitory weights that become patchy over time, and excitatory weights, which become strong between close neighbors. As previously mentioned, this means that the maps produced at each layer will form in such a way that neighbors have similar patterns of activity.

4.7 Inconsistency Detection

Inconsistencies are defined as patterns of activity that have not previously been seen. In other words, if only horizontal lines have been seen in the past, vertical lines would be considered inconsistent. This definition lends itself to a natural way to look for inconsistencies: look for large changes in weights throughout the system. In this system it is the inhibitory weights that are watched over time. The change in excitatory weights or afferent weights could

have also been examined (Brazeau *et al.*, 2003). By looking at inhibitory (or excitetory) weights we examine the amount of change within the V1 and MT maps. Looking at the afferent weights then would correspond to examining the amount of change in the previous map(s) as they relate to a cells receptive field.

A different way to describe this is to look at how much the system is learning. If cells remain active consistently over time, then weight changes will occur fast at first, and slow down over time. This occurs due to the normalization which occurs when calculating the new weights.

Looking for inconsistent events in motion is done in the same way. The amount of weight change is monitored, and jumps in the amount of weight change are indicative of something new being learned.

The most important factor in setting up the system is the way in which it is trained. The next chapter will show that when trained on random input patterns, the system will be prepared to perceive a wide variety of orientations and motions. This is consistent with research which has shown that cats raised with their eyes shut are still able to see when their eyes are later opened. Although a significant result, it is not the desired behavior for a system that is used for determining inconsistencies since it will lead to an inability to see them. Instead, it is important to train DS-LISSOM in the environment that is understood to be the consistent environment. In this way, when changes to that environment (inconsistent events) occur later, the system will respond by adjusting its weights. Again, this is consistent with previously discussed research, which has shown the effects of raising cats in atypical environments.

4.8 Conclusion

The learning system proposed and implemented as DS-LISSOM is a biologically inspired system that is capable of learning both orientation and motion information in a way which is consistent with current knowledge of biological systems. DS-LISSOM further extends previous models by looking at ways to infer that new knowledge is being acquired, and thereby finding inconsistent events. The next section will look at examples of the system at work, and will show that the receptive fields and orientation maps that are formed are biologically accurate.

Chapter 5

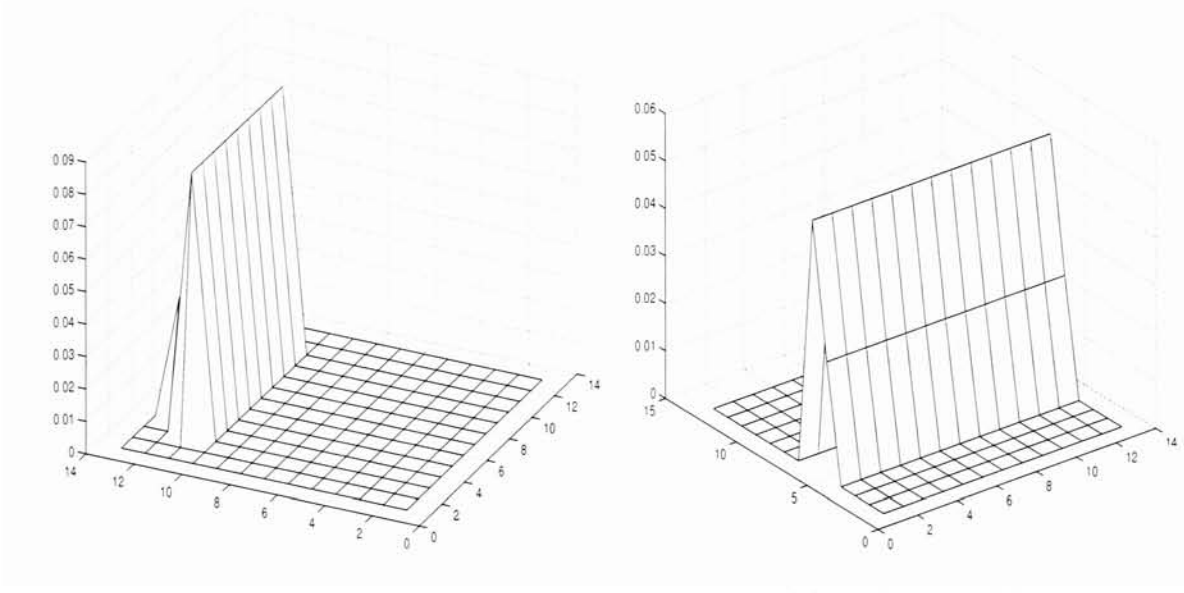
Development of Visual Preferences

This chapter looks at results produced by DS-LISSOM, which was described in the previous section. The results show that it is possible to train the system in such a way that it becomes familiar with both orientation and directions in a general way. Results will further show that when DS-LISSOM is trained on specific input, it will form preferences for only those orientations and directions present within the training set. Finally experiments will show that the system is capable of distinguishing orientations and directions that it has seen before from those it has not seen before.

Each of the experiments were run using the parameter values seen in Appendix B. Using these values DS-LISSOM required 400 megabytes of memory.

5.1 Formation of Selective Neurons

The first, and perhaps most important, result is the formation of cells that are orientation and direction sensitive. Such formation is a direct measure of the capability of the system to learn and retain preferences for visual stimulus.



(a) Edge Selective Cell (b) Bar Selective Cell

Figure 5.1: These are two examples of cells that are formed when a strong edge and bar are present in an image. The result is the formation of edge and bar detectors that activate most strongly when an edge or bar is present in the appropriate part of its receptive field.

5.1.1 Orientation Selectivity

As previously discussed, orientation selectivity first occurs in the V1 layer of the visual system. If DS-LISSOM learns like its biological equivalents then it too should form preferences for orientation in V1. As Figure 5.1 shows, cells can form such preferences. Theses images were obtained by training DS-LISSOM on random dark and light bars. The images show the strength of the weights between the cell and each of the cells in its receptive field as shown in Figure 5.3. Additional figures show input images, their resulting activations within V1 and the orientation selectivity of some of the cells of V1.

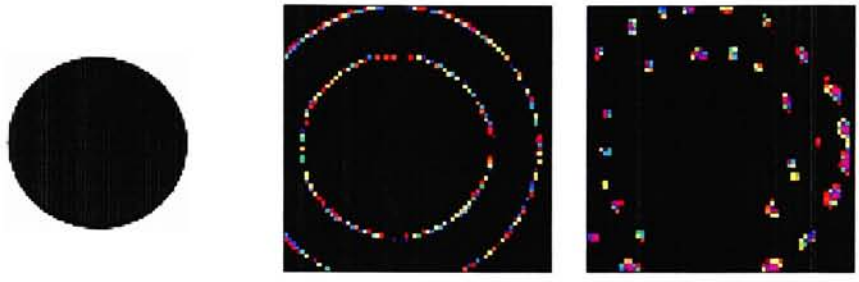
Recalling what was said about the formation of simple cells, Figure 5.4 shows cells that have formed while training DS-LISSOM, and compares them with their simple cell counterpart.

By looking at each of the cells in V1 and determining their orientation preference, an orientation map can also be formed. Figure 5.5 shows one such map. Notice that there are many features present, which is consistent with current research studies.

5.1.2 Motion Selectivity

Motion is learned by combining V1 responses at different moments in time. The result is the formation of cells in V1 that respond to oriented edges moving in a specific direction. Figure 5.6 gives one example of the selectivity of a cell by looking at its afferent weights into each of the V1 layers from different time steps (recall that there are 4 such layers in DS-LISSOM).

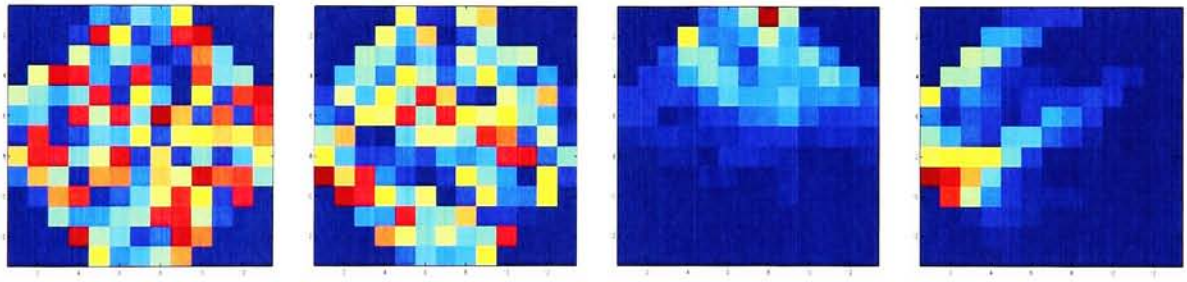
The receptive field for each of the cells in MT is larger then those in V1.



(a) Circle as Input

(b) Early V1 Response

(c) Learned V1 Response



(d) Early Unselective Cell

(e) Early Unselective Cell

(f) Early Selective Cell

(g) Early Selective Cell

Figure 5.2: The first row shows the input image, the response in V1 after only a 10 iterations, the response after 10,000 iterations. The second row shows a number of cells after 10 iterations and after 10,000 iterations.

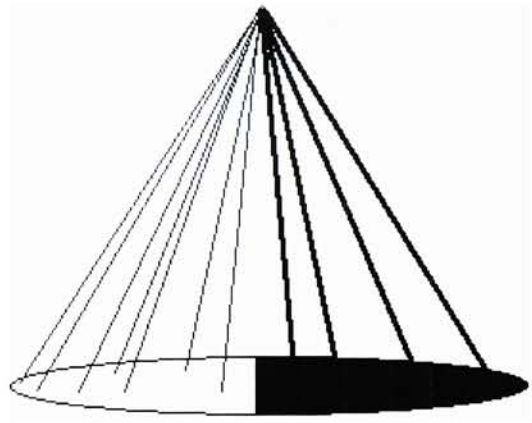
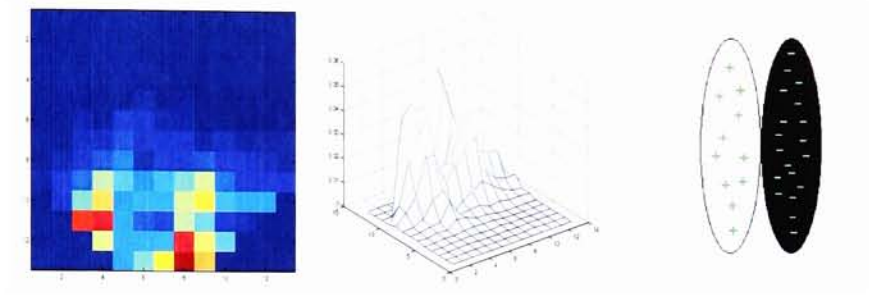


Figure 5.3: The lines represent connections between a cell in V1 and cells in the LGN that are within its receptive field. The lighter lines represent connections with strong weights while darker lines represent connections with weak weights. This particular cell will initially become active if there are a large number of bright cells on its left side and little activity on its right side (indicating the presence of an edge).

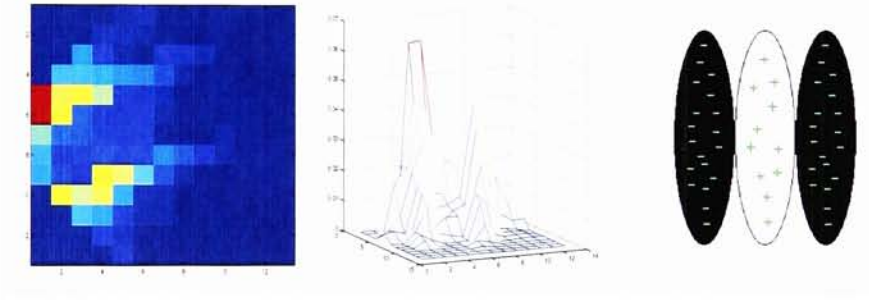
As a result the orientation information is not as important; rather, it is the overall movement of an oriented edge as it moves through the receptive field of a particular cell.

5.2 Finding Inconsistencies

Realizing that DS-LISSOM is capable of learning and developing in the same way that current research tells us the visual system learns and develops leads to a question of whether or not certain aspects of DS-LISSOM can be put to use. Recall the findings of Weisel and the experiments he performed on cats. Such experiments have shown that if a visual system is trained in an atypical environment, such as only seeing horizontal lines, then the visual system



(a) V1 Edge Cell (b) V1 Afferent Weights (c) V1 Simple Cell Equivalent



(d) V1 Bar Cell (e) V1 Afferent Weights (f) V1 Simple Cell Equivalent

Figure 5.4: The first row gives an example of a simple edge cell formed by DS-LISSOM when trained on 10,000 random ellipsoids. The second row gives an example of a simple bar cell, again formed by training the system on random ellipsoids. The first two columns give different views of a cell's afferent weights, which are used to describe its receptive field and when it will become active. The third column shows the equivalent simple cell as it was described in previous chapters. This example helps to show that DS-LISSOM is in fact capable of of learning and forming cells that are sensitive to orientation, as are those observed in biological experiments.

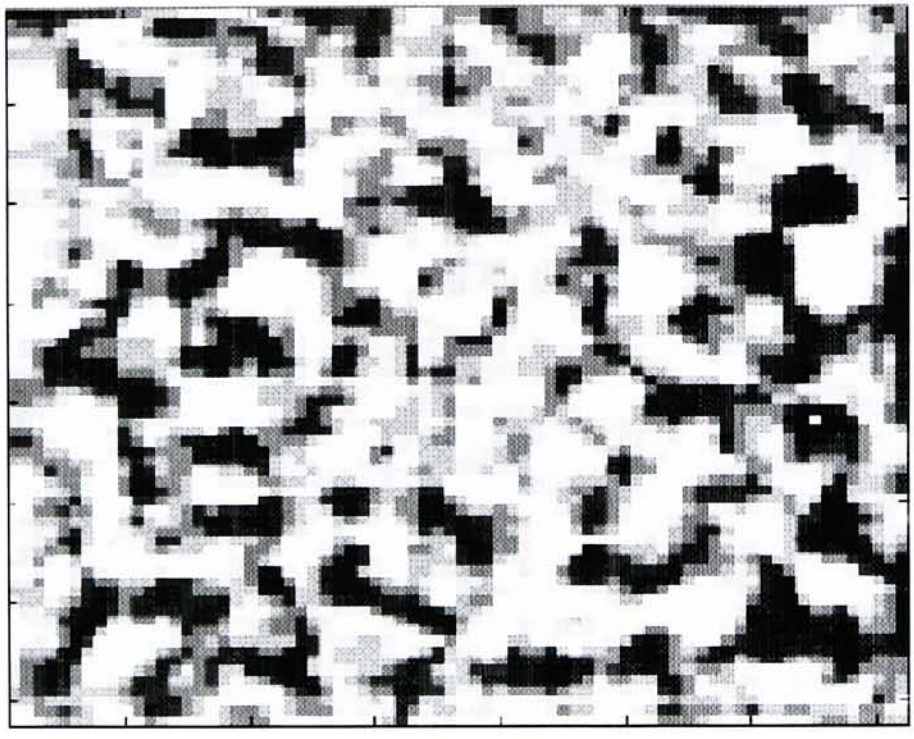
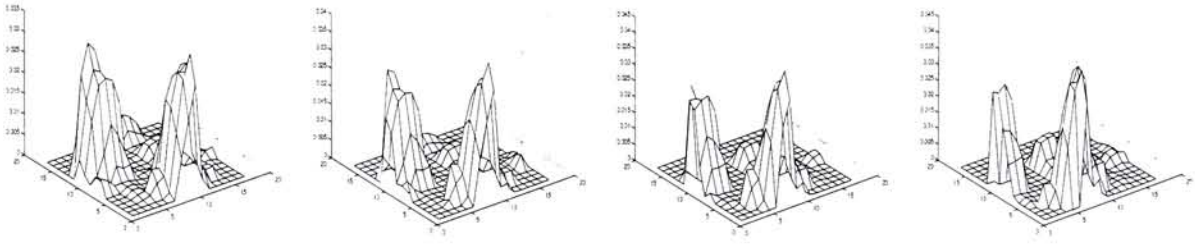


Figure 5.5: This orientation map was formed after training the system for 10,000 iterations on random ellispoid patterns. The darker colors are preferences for horizontal orientations while light colors are more selective of vertical orientations. The result is a map that shows that system has a wide variety of orientation preferences. Similar maps can be formed to show the directional preferences of MT.



(a) Time Step 1 (b) Time Step 2 (c) Time Step 3 (d) Time Step 4

Figure 5.6: Each of the above images represent one period in time and the weights between cells in a particular time period in a cells receptive field to a cell in MT. An MT's receptive field, in other words, not only contains an area at the current time step, but that same area in previous time steps. As shown by the images this allows an MT cell to be most active when and an object appears in the appropriate regions of its receptive field in each time period. The above sequence shows a preference for movement in the rightward direction.

will develop a strong preference for horizontal orientations, while preferences for vertical orientations are weak. This leads to an extention of DS-LISSOM whereby it is used to look for inconsistencies within a particular environment, as described in the previous chapter.

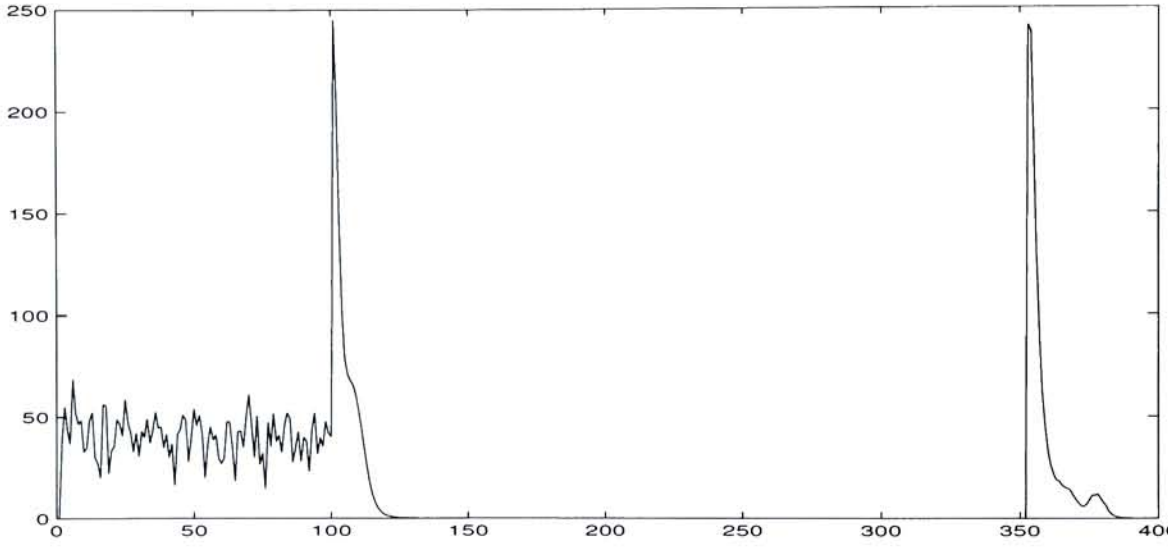
5.2.1 Finding Changes in Orientation

Figure 5.7 shows one example of DS-LISSOM being trained on only horizontal lines and then being placed in an environment with only vertical lines. The result is a huge jump in the amount of weight change in the inhibitory weights of V1. As suggested in chapter 4, this indicates that there is something inconsistent in this new environment. It is important to note that at this time DS-LISSOM does not have the ability to label the inconsistency, rather it can only indicate that one is present.

Figure 5.8 shows the same experiment: placing the system in an environ-

(a) Horizontal Control Image

(b) Vertical Experimental Image



(c) Orientation Response

Figure 5.7: This figure demonstrates the weight changes that occur when the input to the system changes. The first large spike in the plot occurs when DS-LISSOM first sees the horizontal lines. Since at this moment the map is completely random, there are large changes in weight. The second large spike occurs when vertical lines are introduced for the first time to the system. The top two figures show examples of the vertical and horizontal lines that are sent to the system.

ment with only horizontal lines and then changing that environment. This time, however, the system is first trained prenatally on a number of random ellipsoids. Although a large change in weights is observed, the overall amount of change is decreased, indicating that the system has already learned some things about both horizontal and vertical lines.

5.2.2 Finding Changes in Motion

Inconsistencies in motion can be found in the same way. Figures 5.10 and 5.11 show some examples of the types of results achieved by using DS-LISSOM.

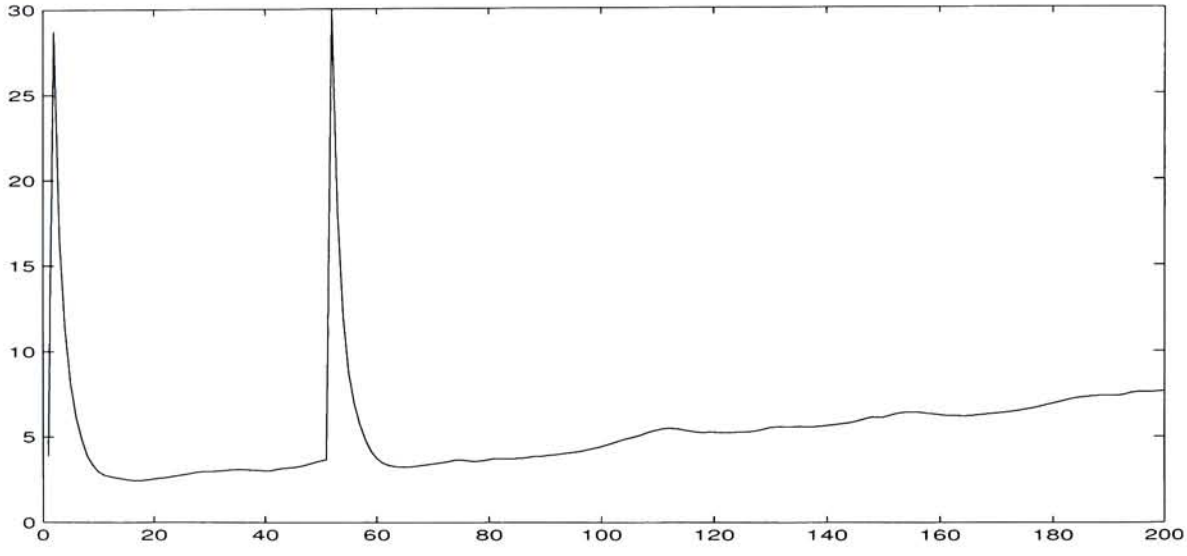
5.2.3 Long Term Memory

When looking for inconsistencies it is also important to think about what an inconsistency at one period means if that same event occurs in the future. Figure 5.2.3 shows that while DS-LISSOM is still in a mode of learning it can adapt to its new environment, while still remembering some of its old environment. If the new stimulus occurs often enough, it will no longer react to this stimulus as though it were inconsistent. At the same time, the original environment is not forgotten as long as it is still reinforced from time to time.

In some cases, this is not the desired result. Instead, it may be the case that any occurrence of an event that did not occur during training should be marked as inconsistent. In these cases the solution is to stop the learning process completely. In this way weights are no longer adjusted; however, the amount of weight change that would have occurred can be recorded and used to determine if an event has occurred in the past.

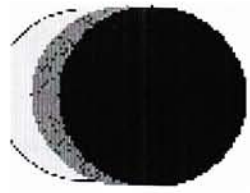
(a) Horizontal Control Image

(b) Vertical Experimental Image



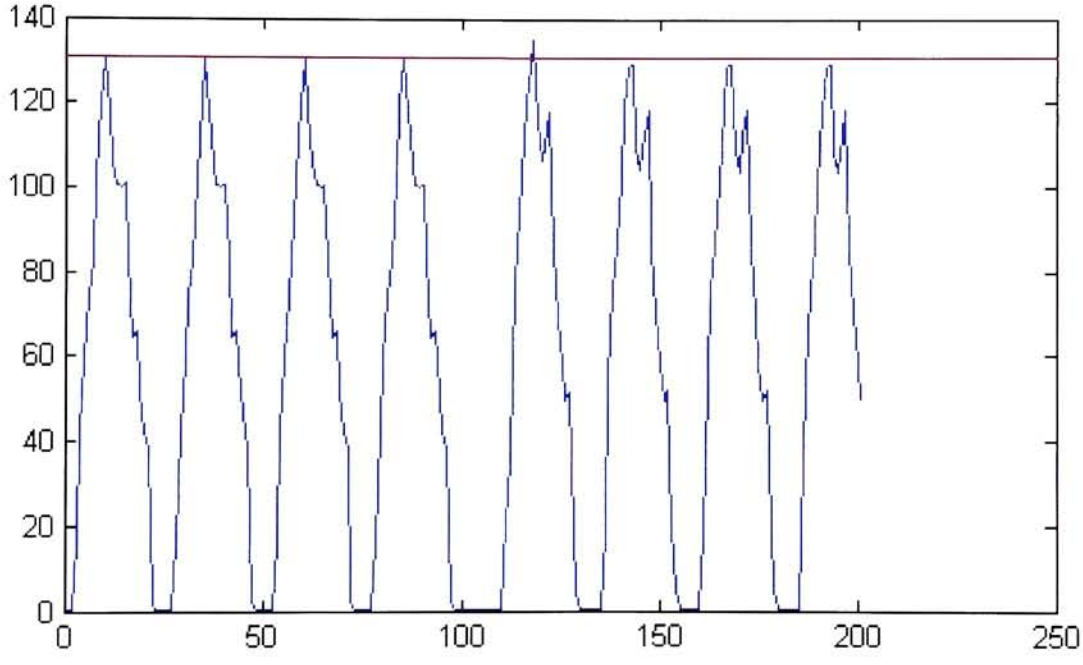
(c) Orientation Response

Figure 5.8: Unlike 5.7 this experiment was performed on DS-LISSOM after it had been prenatally trained on random ellipsoids. Although learning again occurs, it is important to realize that the relative amount of change in the maps is significantly reduced. Learning still must occur since the system has never seen horizontal and vertical lines presented in this way.



(a) Circle Moving Right

(b) Circle Moving Left

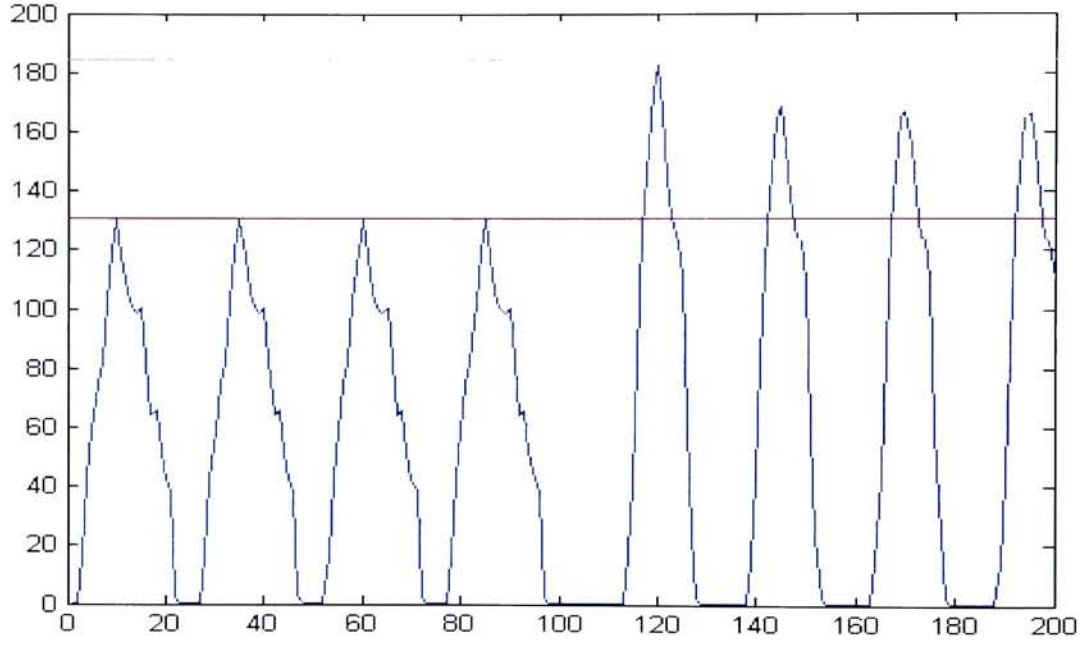


(c) Plot of Weight Changes

Figure 5.9: In this example, no pretraining occurred. In this way, the system became oriented to seeing only a few specific orientations. The top row shows the type of input images that were sent to the system. The first 900 iterations were input of a circle moving from the left to the right. After that the circle changed direction, moving from the right to the left. With this change in direction came a change in the amount the inhibitory weights were being adjusted.

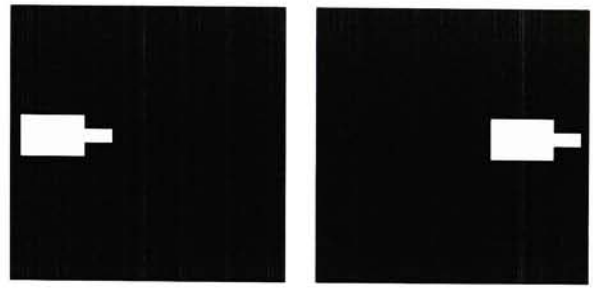
(a) Circle Moving Right

(b) Circle Moving Up and Right

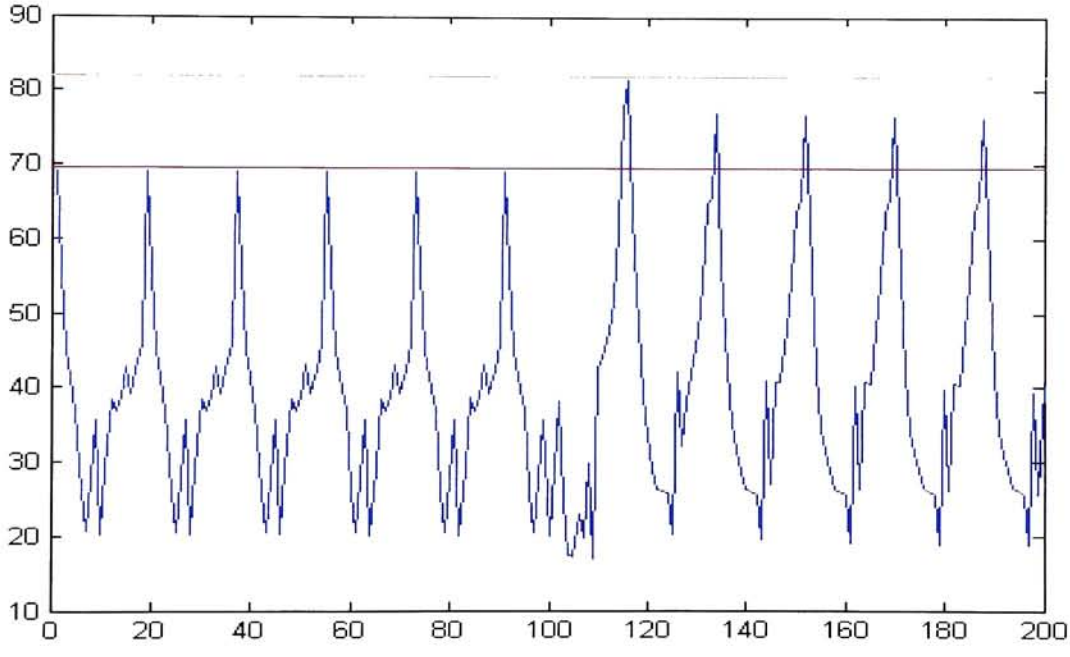


(c) Plot of Weight Changes

Figure 5.10: Like 5.9, this example was performed on DS-LISSOM with no pre-training. The top row shows the types of input images that were sent to the system. Images of a circle moving from the left to the right were displayed for the first 900 iterations. Next, images of a circle moving up and to the right were shown. The increase in weight change coincides with the change in direction.

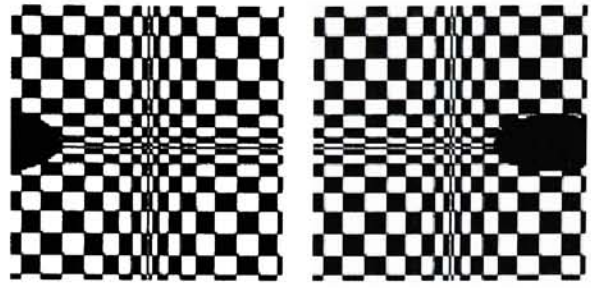


(a) Boxes Moving Right (b) Boxes Moving Left

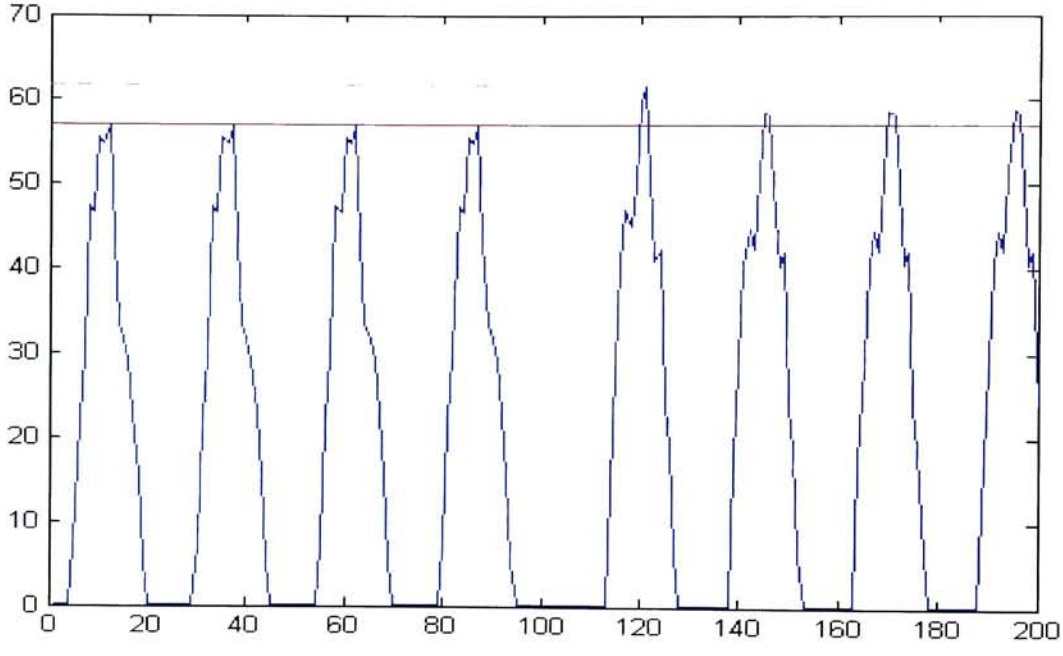


(c) Plot of Weight Changes

Figure 5.11: The detection of movement occurs not only with circles, but in the plot, with boxes as well.



(a) Circle Moving Right (b) Circle Moving Left



(c) Plot of Weight Changes

Figure 5.12: DS-LISSOM is also able to distinguish complex backgrounds from the moving objects. The above plot shows the detection of an ellipse's change of direction, even when a complex background exists.

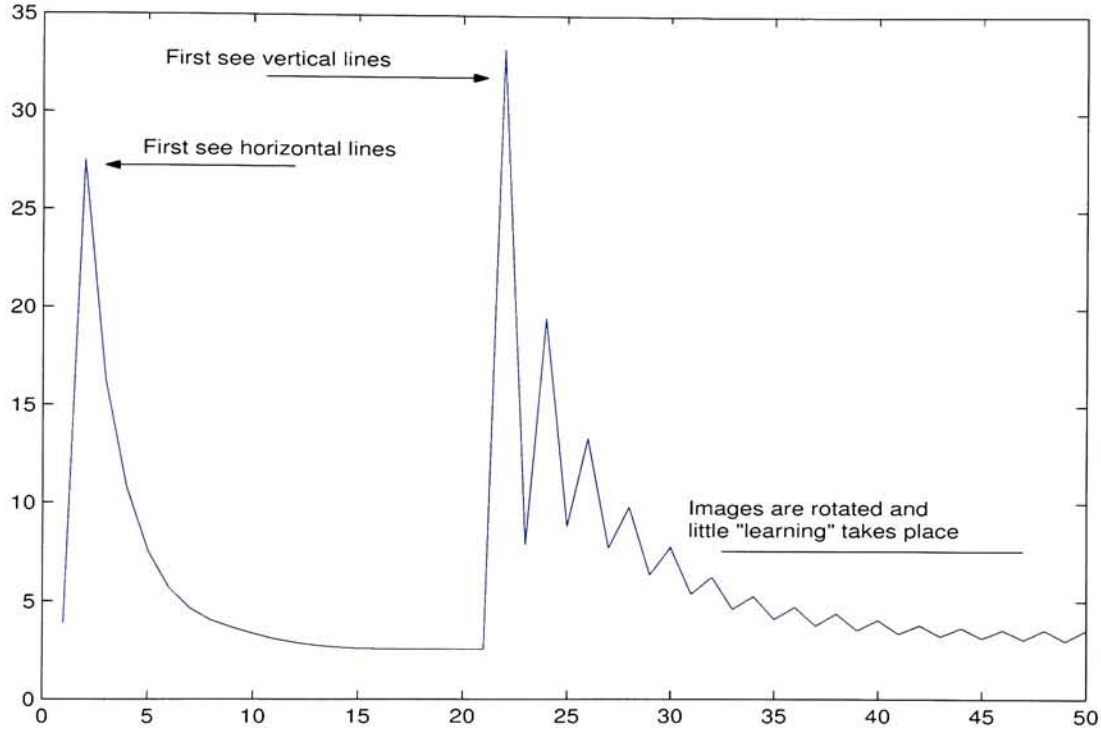


Figure 5.13: Once DS-LISSOM has seen an image, it is capable of retaining that knowledge. In the above sequence DS-LISSOM was shown horizontal lines 20 times in a row, followed by vertical and horizontal lines being displayed one after the other. As the plot shows DS-LISSOM reacts less strongly.

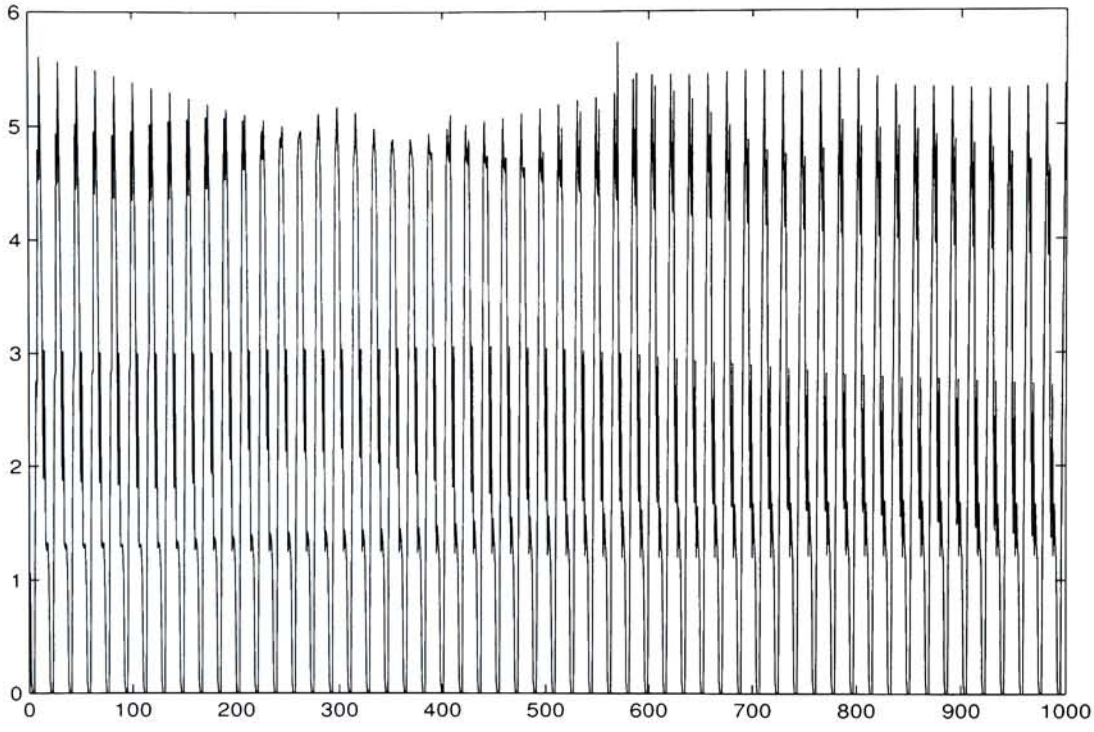


Figure 5.14: The above plot shows that once DS-LISSOM is trained it is already capable of perceiving a number of different directional motions. The amount of weight change is minimal, since many parts of the map are already tuned to this directional motion.

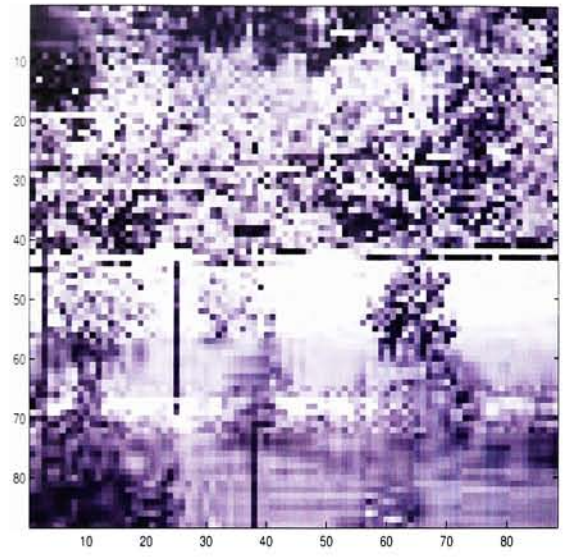
5.2.4 Working with Natural Images

The following experiment was conducted as follows:

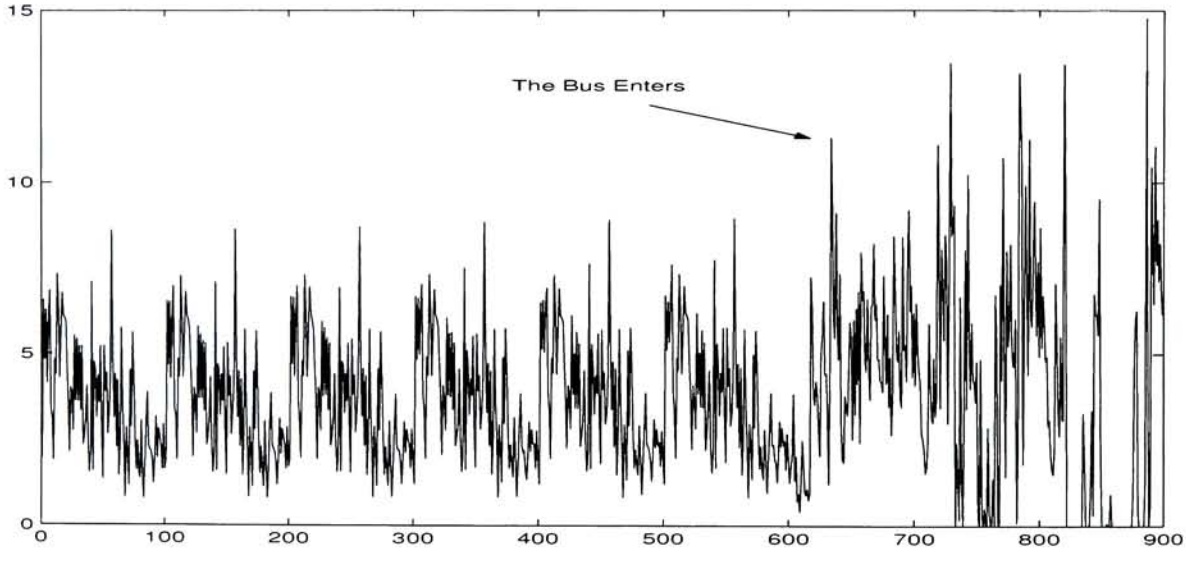
1. Train the system on random ellipsoid patterns (10,000 iterations).
2. Place the system in an environment where it will only see cars moving from the right to the left (2,000 iterations).
3. Change the environment by showing a bus which moves from the left to the right of the image.

The results show that it is possible for the system to recognize the difference. What is important to note is the fact that the system was trained on a large number of random ellipsoids initially. This meant that the typical motion of objects in the background were already learned. Large movements, such as that of the cars, were learned over time. Finally, when the environment changed and a bus moved in the opposite direction, the map reorganized to compensate, and thus the amount of weight change increased. Figure 5.15 shows some of the results from this experiment.

The type of training has the completely opposite effect when working with simple images. Figure 5.14 shows that when a box changes direction after training on random ellipsoids, there is little change in weight. Most likely this is because the directional motion is very simple and is already built into the system through prenatal development. Recall that the same result was seen when looking at the change in V1 weights when working with horizontal and vertical lines displayed both with training and without training (Figures 5.7 and 5.8).



(a) Input Image



(b) Weight Plot

Figure 5.15: The input image is of two roads. The top road is the only one where motion from cars occurs. At first, cars will only pass from the right to the left. After some time a bus moves from the left to the right. The plot shows the change in learning that occurs when the bus enters. Because the system was first trained on random elipsoids, much of the other movement, such as the movement in the trees, does not effect the results.

One problem with the system as it is set up now are limitations with memory. The above examples required over 400 megabytes of memory to run. If the size of the input image is increased, the amount of memory required increases exponentially. Matlab 6.5 running under Windows XP is not capable of using more then 1 Gigabyte of memory. As a result, natural images need to be small, and those images which are larger have a poor resolution. Therefore, as it is currently implemented, the ability for DS-LISSOM to work with natural images is limited.

5.3 Conclusion

DS-LISSOM is a general purpose visual learning system that is capable of learning both orientation and direction information. The proceeding experiments have shown that DS-LISSOM is capable of forming simple cells like those found in the primary visual cortex, and of producing cells that are directionally selective like those found in the MT area. Finally, experiments show that it is possible to exploit the learning process of DS-LISSOM in order to detect inconsistencies that occur within a specific environmental setting.

Chapter 6

DS-LISSOM: A Users Guide

During the development and testing of DS-LISSOM, a graphical environment was developed and is described in this chapter. In general the system allows for a user to train the system on random ellipsoids for a specified amount of time, after which it will be trained on a specific environment. The user can also ask for the system to be placed into a second environment which may, or may not, be inconsistent.

6.1 Explanation of the GUI

In 6.2 each of the parts of the GUI are labeled, and their function is detailed here.

1. The input image is the image as it is input into the system.
2. LGN Response is the response that the LGN sees once the input image passes through the Retinal Ganglion Cells.
3. The Orientation Response is the response of V1. This image shows the activity that is occurring within the V1 layer.

4. The Motion response represents pixels in V1 only if their activity has changed since the last period in time.

5. The MT Response is the activity that occurs in the MT layer.

6. The MT+ Response represents activity changes between two time periods in MT. It is currently my belief that such information can be useful in finding more complex motion such as contractions and expansions, and may be used to look for self-motion.

7. The V1 Responses plot will show up to the last 300 weight changes. The horizontal axis is the time when the weight measure was taken. The vertical axis represents the amount of change that occurred throughout the entire V1 map.

8. The MT Responses plot is similar to that of the V1 Responses plot, except that it measures changes that occur within the MT layer.

9. The slide bar allows a user to select the number of training iterations. Each training iteration is made up of one image of random ellipsoids. Typically it takes about 10,000 training iterations to fully prepare the system for a wide range of orientations and directional motions.

10. The number of control iterations allows the user to place the system in one environment for a while, and then switch it to a second experimental environment.

11. The control environment in which the system will first be placed.

12. The experimental environment.

13. Allows the user to load a previously run experiment. This is extremely useful since DS-LISSOM can start running from a previously saved training set, saving a great deal of time.

14. This button allows the user to save the correct state of DS-LISSOM, to be loaded at a later time.

15. This button will start DS-LISSOM.

16. This button will stop DS-LISSOM from running. Pressing the Start Training button will then start DS-LISSOM from this point.

17. This button will clear all of the current data from DS-LISSOM so that a new experiment can be run.

18. This button allows the user to quit.

6.2 A simple example

This section describes a simple example of running DS-LISSOM given that a trained version of DS-LISSOM is already available.

1. Start DS-LISSOM by running VisionSystem from the command window of Matlab.

2. Click on "Load Trained Set". This will pop up a file browser and will allow you to select a previously saved dataset. One provided in the DS-LISSOM distribution is 10000.mat.

3. Select the number of training iterations that you would like to take place. Since a trained set was already loaded this can be set to 0.

4. Select the number of training iterations to take place. Lets select 20 for example.

5. Select the control group. A number of them are listed in the pull down box.

6. Select the experimental group. Again, a number of them are already provided.

7. Press the "Start Training" button. This will start DS-LISSOM.

8. If you want to save data at any point in time, press the "Save Current State" button.

9. To quit press the "Quit" button.

Matlab does not allow for multithreading so it is often the case that the GUI seems unresponsive. For this reason there is a 2 second delay between each iteration, in which the user is able to press any button they desire.

6.3 Using Different Data

DS-LISSOM's GUI currently does not explicitly allow for a user to select their own control and experimental data sets. It is possible to overwrite the files in a particular data set in order to achieve this goal. Image file names are specified by number in order of the time that they appear. If you want to use a video sequence it must be in AVI format.

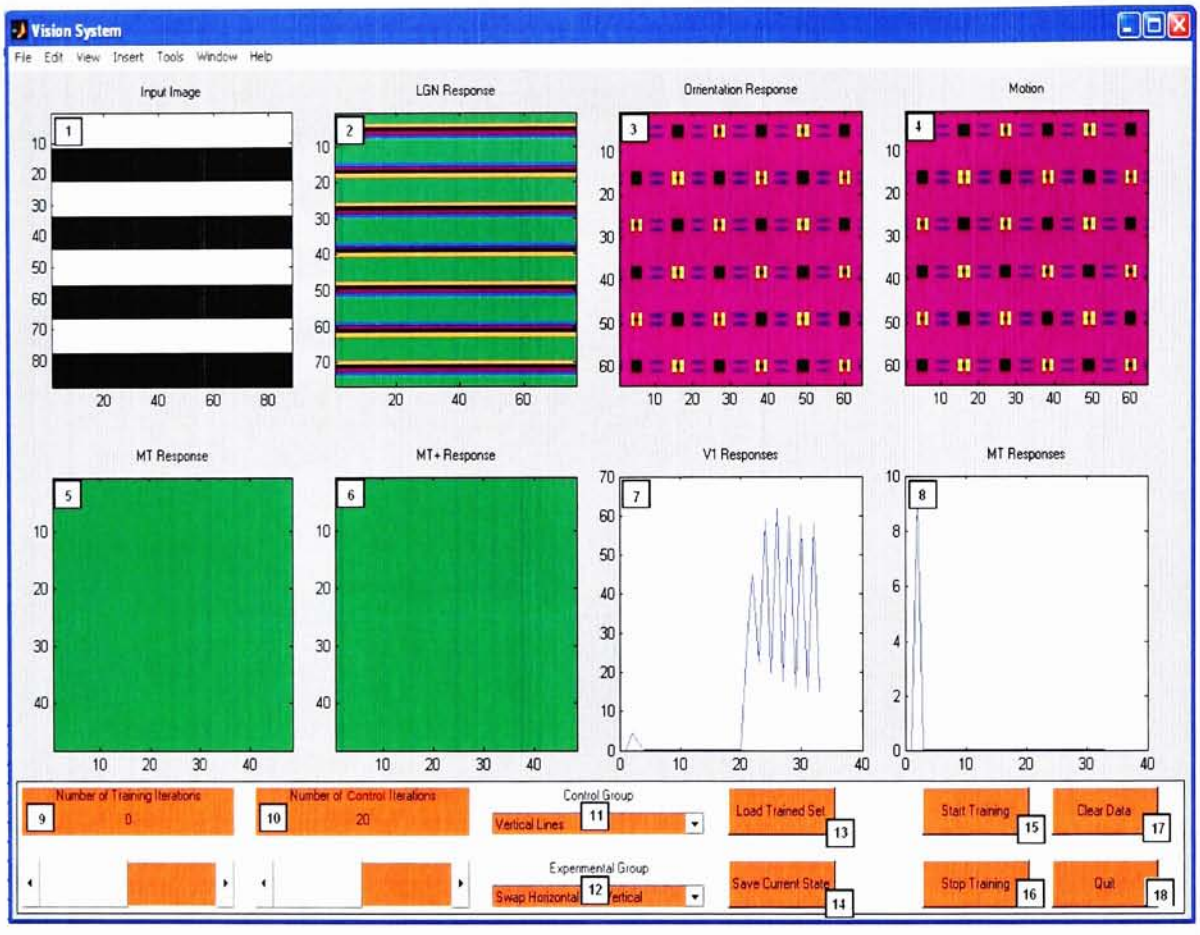


Figure 6.1: A screenshot of DS-LISSOM's graphical user interface

Chapter 7

Conclusion and Future Work

This thesis has investigated the possibility of creating a system whose components are based on biological equivalences. It has been shown that DS-LISSOM is biologically consistent not only in construction but function. Results have also shown that the way in which DS-LISSOM acquires information can be exploited to find inconsistencies within images. These results have been shown to work not only with the most simple of images but DS-LISSOM is also capable of finding inconsistencies within natural images.

Future work aims at both expanding upon the current system and using the system in a number of applications, as described in the following sections.

7.1 Modeling More of the Visual System

This thesis has developed a system that can describe the formation of specific cell structures within the visual system, and has focused on the V1 and MT layers. Even though DS-LISSOM has accomplished its goals, we still desire to produce a more extensive model of the visual system.

7.1.1 More about V1

Some research has shown that V1 is directionally selective, and work has begun on modeling this behavior (Bednar & Miikkulainen, 2003). Adding this behavior to V1 could potentially affect how well higher cortical areas are capable of learning the appropriate information. DS-LISSOM at the moment estimates this effect this has by only passing information to higher levels if some change in activation has occurred.

7.1.2 Modeling MT

MT is the first layer that is both selective of direction and velocity. This thesis has looked at the MT layer only in terms of its directional selectivity but future experiments should be performed to show that the MT layer, as implemented in DS-LISSOM, is selective of speed. There are a few questions that need to be considered when looking at speed. For example, DS-LISSOM uses discrete time steps that never change. It may be that such a simple approach is not adequate enough when looking at velocity. It may also be the case that using information from only four points in time is insufficient for learning velocity.

7.1.3 Modeling MST

Information passes through MT and into MST and a number of other smaller regions which do further motion processing. These areas can process more complex motions and also fuse information about overall motion (such as eye movements and optic flow) into the system. It would be a natural extension to attempt to model the MST area.

7.2 Finding Inconsistent Events

As implemented, DS-LISSOM only gives the user information about the amount change in weights. The next step in this direction is to actually label images that have been found to be inconsistent. Not only would it be useful to label the image, but it would also be useful to label the parts of the image which are thought to be inconsistent. This could be accomplished using the following:

1. Process an image or a series of images.
2. If an image is found to be inconsistent then do the following recursively.
    - Break the image at this time step and the two previous time steps into 4 regions.
    - Check for the relative amount of weight change in each region from one time step to the next.
    - Continue to recursively check those regions which are inconsistent until no one region is more inconsistent then the others. These 4 regions now make up the inconsistent part of the overall image.

Appendix A

Source Code

The following sections are the source of DS-LISSOM implemented in matlab.

A.1 Vision System

```
1
2
3       %Declare All Global Variables
4       global original_h;
5       global lgn_h;
6       global orrientation_h;
7       global color_h;
8
9       global training_pathname;
10      global detect_pathname;
11      global training_filename;
12      global detect_filename;
13
14
15      %Create and run the GUI interface
16      CreateGUI
```

A.2 GUI Creation

```
1  clear;
2
3  % This file will initialize and run our vision system from start to finish.
```

```
4  % This GUI will provide a user to train a system from scratch, view different
5  % outputs from the system, and will allow a previously trained network be loaded
6  % and run.
7  %
8  % For better efficiency it may be neccessary for training to take place as a
9  % seperate step.  However, this GUI should make it easy for someone new to the
10 % system to get started.
11 %
12
13 %Declare All Global Variables
14
15 % GUI Variables
16 global Input_h;
17 global LGN_h;
18 global V1_h;
19 global difference_V1_h;
20 global MT_h;
21 global difference_MT_h;
22 global up_h;
23 global down_h;
24 global right_h;
25 global left_h;
26 global MTGraph_h;
27 global V1Graph_h;
28
29 % Files
30 global training_pathname;
31 global detect_pathname;
32 global training_filename;
33 global detect_filename;
34
35
36 % Image Sets
37 global controlSet;
38 global expSet;
39
40 % Used for movement through-out
41 global training;
42 global stopEverything;
43 global trainingIterations;
44 global controlIterations;
45 global saveFile;
46 global loadFile;
47 global startOver;
48 global start;
49
50
51 %
```

```
% Initialize needed variables
%

controlSet = 1;             % Set the default control set
expSet = 1;                 % Set the default experimental set
training = 0;               % Don't train yet
stopEverything = 0;         % Set on quit
trainingIterations = 100;   % How long to train
controlIterations = 100;    % How long should the control image(s) be shown
saveFile = 0;               % Save the file now
loadFile = 0;               % Load a saved file
startOver = 1;              % Clear everything
start = 0;                  % Begin initializing and
                            % load any data if needed


training_pathname = '';             % Default path is where the program is
                                    % started.
training_filename = 'training.mat'; % The save name will be training.mat
                                    % followed by the iteration number



    %Create a Figure
    figpos=[70 00 1100 700];
    fig = figure('Name','Vision System','NumberTitle','off', ...
                 'Pos',figpos,'Units','pixels');


    %Create a frame for any options we may have!
    uicontrol('Style','frame','Pos',[5 5 figpos(3)-10 110]);


    %*********************************************************************
    %
    % Create and display the axes for each of the displayed results
    %
    %*********************************************************************

    %Create Axes for the original image
    uicontrol('Style','text','String','Input Image', 'Pos', [40 670 225 20], ...
              'BackgroundColor', get( fig, 'color' ) );
    Input_h = axes('Units','pixels','Pos',[40 440 225 225],'Box','off', ...
                   'Tag','Original_image');


    %Create Axes for the LGN Response
    uicontrol('Style','text','String','LGN Response', ...
```

```
100             'Pos', [305 670 225 20], ...
101             'BackgroundColor', get( fig, 'color' ) );
102     LGN_h = axes('Units','pixels','Pos',[305 440 225 225],'Box','off', ...
103                     'Tag','LGN_Response');
104
105
106     %Create Axes for the Orientation Response
107     uicontrol('Style','text','String','Orrientation Response', ...
108             'Pos', [570 670 225 20], ...
109             'BackgroundColor', get( fig, 'color' ) );
110     V1_h = axes('Units','pixels','Pos',[570 440 225 225],'Box','off', ...
111               'Tag','Orrientation_Response');
112
113
114     %Create Axes for the Orientation Response
115     uicontrol('Style','text','String','Motion', 'Pos', [835 670 225 20], ...
116             'BackgroundColor', get( fig, 'color' ) );
117     difference_V1_h = axes('Units','pixels','Pos',[835 440 225 225], ...
118                         'Box','off', ...
119                         'Tag','Orrientation_Response');
120
121
122     %Create Axes for the Color Response
123     uicontrol('Style','text','String','MT Response', 'Pos', [40 370 225 20], .
124             'BackgroundColor', get( fig, 'color' ) );
125     MT_h = axes('Units','pixels','Pos',[40 140 225 225],'Box','off', ...
126                   'Tag','Color_Response');
127
128
129     %Create Axes for the Color Response
130     uicontrol('Style','text','String','MT+ Response', ...
131             'Pos', [305 370 225 20], ...
132             'BackgroundColor', get( fig, 'color' ) );
133     difference_MT_h = axes('Units','pixels','Pos',[305 140 225 225], ...
134                         'Box','off', ...
135                         'Tag','Color_Response');
136
137
138     %Create Axes for the Color Response
139     uicontrol('Style','text','String','V1 Responses', ...
140             'Pos', [570 370 225 20], ...
141             'BackgroundColor', get( fig, 'color' ) );
142     V1Graph_h = axes('Units','pixels','Pos',[570 140 225 225],'Box','off', ...
143                     'Tag','Color_Response');
144
145
146     %Create Axes for the Color Response
147     uicontrol('Style','text','String','MT Responses', ...
```

```
            'Pos', [835 370 225 20], ...
            'BackgroundColor', get( fig, 'color' ) );
MTGraph_h = axes('Units','pixels','Pos',[835 140 225 225],'Box','off', ...
                'Tag','Color_Response');


%
% Set the colormap.  For the system's images the best is rainbow.  For
% input images the best would be gray.  I have not figured out how
% to use multiple colormaps in a single figure.
%
colormap( 'rainbow' );
%colormap( HSV );

%**************************************************************************
%
% Create Buttons, Lists etc. for human interaction
%
%**************************************************************************

%Create Buttons for User Interaction
uicontrol('Style','push','String','Quit', ...
          'Pos',[970 10 100 40],'Callback',[' training = 0;',...
                                            ' stopEverything = 1;',...
                                            ' pause(5);',...
                                            ' close(gcf);'], ...
              'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);

uicontrol('Style','push','String','Clear Data', ...
          'Pos',[970 70 100 40],'Callback','startOver = 1;', ...
              'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);

uicontrol('Style','push','String','Start Training', ...
          'Pos',[850 70 100 40],'Callback','training = 1;, start = 1;', ...
              'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);

uicontrol('Style','push','String','Stop Training', ...
          'Pos',[850 10 100 40],'Callback','training = 0; start = 1;', ...
              'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);


%This is the slider for setting the training iterations
slider.l = uicontrol('Style','text','Pos',[10 90 200 20], ...
             'String','Number of Training Iterations', ...
             'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);

slider.n = uicontrol('Style','text','Pos',[10 70 200 20], ...
             'String',num2str(trainingIterations),...
```

```
196                 'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);
197
198     slider.s = uicontrol('Style','slider','Pos',[10 10 200 40],'Min',0,...
199                 'Max',10000,'Value',trainingIterations,...
200                 'SliderStep',[.01 1],'BackGroundColor',[1 0.5 0],...
201                 'ForeGround', [0 0 0],...
202                 'CallBack', ['trainingIterations=get(slider.s,''value'');', ...
203                              'set(slider.n,''String'',trainingIterations);'] );
204
205
206
207     %This is the slider for setting the control iterations
208     slider2.l = uicontrol('Style','text','Pos',[230 90 200 20], ...
209                 'String','Number of Training Iterations', ...
210                 'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);
211
212     slider2.n = uicontrol('Style','text','Pos',[230 70 200 20], ...
213                 'String',num2str(trainingIterations),...
214                 'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);
215
216     slider2.s = uicontrol('Style','slider','Pos',[230 10 200 40],'Min',0,...
217                 'Max',10000,'Value',trainingIterations,...
218                 'SliderStep',[.01 1],'BackGroundColor',[1 0.5 0],...
219                 'ForeGround',[0 0 0],...
220                 'CallBack', ['controlIterations=get(slider2.s,''value'');', ...
221                              'set(slider2.n,''String'',controlIterations);'] );
222
223
224
225     %Allow a user to select what images to train on postnatally and
226     %what environment to subsequently place DS-LISSOM into
227
228     control.l = uicontrol('Style','text','Pos',[450 90 200 20], ...
229                 'String', 'Control Group' );
230     control.p = uicontrol('Style','popupmenu','String',...
231                                         ['Box moving left to right|',...
232                                         'Box moving right to left|',...
233                                         'Cars|',...
234                                         'Bus|',...
235                                         'Horizontal Lines|',...
236                                         'Vertical Lines|',...
237                                         'Circle'],...
238                 'CallBack', ['controlSet=get(control.p,''value'');'], ...
239                 'Pos',[450 70 200 20], ...
240                 'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);
241
242     exp.l = uicontrol('Style','text','Pos',[450 30 200 20], ...
243                 'String', 'Experimental Group' );
```

```
244     exp.p = uicontrol('Style','popupmenu','String',. .
245                                             ['Box moving left to right|',...
246                                             'Box moving right to left|',...
247                                             'Cars|',...
248                                             'Bus|',...
249                                             'Horizontal Lines|',...
250                                             'Vertical Lines|',...
251                                             'Swap Horizontal and Vertical'],...
252                 'CallBack', ['expSet=get(exp.p,''value'');'], ...
253                 'Pos',[450 10 200 20], ...
254                 'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);
255
256
257     uicontrol('Style','push','String','Load Trained Set', ...
258               'Pos',[670 70 100 40], ...
259               'Callback',['[pathname,filename] = LoadData, ', ...
260                ' copyfile( strcat(pathname,filename), ',...
261                ' strcat(training_pathname,training_filename) );,', ...
262                ' start = 0; loadFile = 1; startOver = 1;' ], ...
263                   'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);
264
265     uicontrol('Style','push','String','Save Current State', ...
266               'Pos',[670 10 100 40], ...
267               'Callback', ['[pathname,filename] = SaveData, saveFile = 1'], ...
268                   'BackGroundColor',[1 0.5 0],'ForeGround',[0 0 0]);
269
270
271     %
272     % Start DS-LISSOM.  I don't know how to spawn a seperate thread so this
273     % will run in an infinite loop and callbacks will set global variables
274     % that can change the execution path.
275     %
276     StartTraining( start, training_pathname, training_filename );
```

A.3 DS-LISSOM

```
1   %
2   % This is the main program.  It consists of a number of large loops that
3   % will look at variables set by the gui to determine how it should run.
4   % This follows the system described in my thesis.
5   %
6
7   function StartTraining( first_time, pathname, filename )
8
9       % GUI Variables
```

```
10      global Input_h;
11      global LGN_h;
12      global V1_h;
13      global difference_V1_h;
14      global MT_h;
15      global difference_MT_h;
16      global up_h;
17      global down_h;
18      global right_h;
19      global left_h;
20      global MTGraph_h;
21      global V1Graph_h;
22
23      % Files
24      global training_pathname;
25      global training_filename;
26
27      % Image Sets
28      global controlSet;
29      global expSet;
30
31      % Used for movement through-out
32      global training;
33      global stopEverything;
34      global trainingIterations;
35      global controlIterations;
36      global saveFile;
37      global loadFile;
38      global startOver;
39      global start;
40
41
42      %
43      % Continue as long as the EXIT button has not been pressed.  This is
44      % caught by watching the stopEverything variable.
45      %
46
47      while( stopEverything == 0 )
48
49          % If start is false then pause for 2 seconds and go back to the
50          % top of the while loop.
51          if( start == 0 )
52              pause(2);
53              continue;
54          end;
55
56          %
57
```

```
% Start the system
%

disp( 'Starting DS-LISSOM' )



%**********************************************************************
% Initialize Data - Only if we are starting over
%

if( startOver == 1 )

    %
    % Start from scratch unless a file to start from was loaded
    %

    if( loadFile == 0 )
        MT_NOT_SET = true;
        this_rounds_image = 1;

        % The following must be uncommented to make use of an USB camera
        % vcapg2;


        % All of this is taken from the appendix and from
        % chapter 4 and 5 of Bednars PhD Thesis.  He references a
        % number of other papers that describe a large set of tests
        % that were performed to determine the optimal values and
        % describes how to scale these values.
        %
        % Basically, if something is a "subscript", it's marked with
        % an underscore.  For example, r<sub>A</sub> for the afferent
        % radius is marked by r_A.  However, be aware that SOME
        % VALUES DO INDEED CHANGE IN THE COURSE OF DS-LISSOM
        % especially the gamma values and the r_A from receptors to LGN.

        s_a = 2;                              % Scaling factor
        r_A_o = 6.5;                          % Original radius
        r_A = 6.5;                            % radius V1
        r_MT = 8.5;                           % MINE: radius MT
        r_E_o = 19/2;                         % radius E weights
        r_I_o = 47/2;                         % radius I weights
        MT_d = s_a * 24;                      % MINE: Size of MT
        %N_d = s_a * 24;                      % Bednars V1 size
        N_d = MT_d + 2 * ( r_MT   0.5 );      % MINE: Size of V1
        LGN_d = N_d + 2 * ( r_A_o - 0.5 );    % Size of LGN
        R_d = LGN_d + 2 * ( r_A   0.5 );      %
```

```
106                 r_E_i = floor( N_d / 10 ) + 0.5;    % radius E weights start
107                 r_E_f = floor( max( 1.5, N_d/44 ) ) + 0.5;  % E weights finish
108                 r_E = r_E_i;                        % set E weights radius
109                 r_I = floor( N_d / 10 ) + 0.5;      % set I weights radius
110                 s_w = (r_A_o + 0.5)/r_A;            % scale weight radius
111                 A = 7.5 / s_w;                      % Size of DoG
112                 B = 1.5 / s_w;                      % Size of DoG
113
114                 %
115                 % Sigma and Gamma values for each set of weights
116                 %
117                 sigma_c = 0.5 / s_w;
118                 sigma_s = 4 * sigma_c;
119                 sigma_A = r_A / 1.3;
120                 sigma_E = 0.78 * r_E;
121                 sigma_I = 2.08 * r_I;
122                 gamma_A = 1;
123                 gamma_E = 0.9;
124                 gamma_I = 0.9;
125                 gamma_E_MT = 0.9;
126                 gamma_I_MT = 0.9;
127                 gamma_N = 0;
128                 gamma_A_MT = gamma_A;
129                 gamma_N_MT = gamma_N;
130
131                 %
132                 % Scaling factors
133                 %
134                 s_d = 1;
135                 s_t = 1 / s_d;
136                 n_A = 1;
137
138                 %
139                 % Learning rates
140                 %
141                 alpha_A_i = 0.0070 / ( n_A * s_t * s_d );
142                 alpha_E   = (0.002 * r_E_o^2) / ( s_t * s_d * r_E^2 );
143                 alpha_I   = (0.00025 * r_I_o^2) / (s_t * s_d * r_I^2 );
144
145
146
147                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
148                 % The weights.  Basically in the format weight( from, to ).
149                 % The initial weights are RANDOMIZED for afferent radii from
150                 % LGN to V1.  All other weights are done by GAUSSIAN.
151                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
152
153                 % Weights from retina to LGN   ie: LGN's afferent weight
```

```
154 % mu_aff_LGN_OFF is simply a negative of mu_aff_LGN_ON.
155 mu_aff_LGN  = zeros( r_A*2,   r_A*2,   LGN_d*LGN_d );
156
157 % Weights from LGN to V1.  There are 2 channels: on and off
158 % which Bednar was able to show makes a difference
159 mu_aff_ON_V1  =  rand( r_A_o*2, r_A_o*2, N_d*N_d  );
160 mu_aff_OFF_V1 =  rand( r_A_o*2, r_A_o*2, N_d*N_d  );
161
162 % Weights from V1 to MT.  Temporal effect start taking place
163 % here so I keep 4 "frames" of information alive at a time
164 mu_aff_1_MT = rand( r_MT*2, r_MT*2, MT_d*MT_d  );
165 mu_aff_2_MT = rand( r_MT*2, r_MT*2, MT_d*MT_d  );
166 mu_aff_3_MT = rand( r_MT*2, r_MT*2, MT_d*MT_d  );
167 mu_aff_4_MT = rand( r_MT*2, r_MT*2, MT_d*MT_d  );
168
169
170 % These are the weights for Face Processesing which
171 % I am not going to do
172 % mu_aff_FSA = rand( N*N, r_A*2, r_A*2 );
173
174
175 % Excitetory/Inhibitory lateral connections
176 % in V1
177 E_V1 = zeros( N_d*N_d, r_E*2, r_E*2 );
178 I_V1 = zeros( N_d*N_d, r_I*2, r_I*2 );
179
180
181 % Excitetory/Inhibitory lateral connections in MT
182 E_MT = zeros( MT_d*MT_d, r_E*2, r_E*2 );
183 I_MT = zeros( MT_d*MT_d, r_I*2, r_I*2 );
184
185
186 % Excitetory/Inhibitory lateral connections
187 % in Face Processing.  They are only here to
188 % be complete in terms of Bednars model
189 %E_FSA = zeros( N*N, r_E, r_E );
190 %I_FSA = zeros( N*N, r_I, r_I );
191
192
193 % Binary circle masks.  It may be possible to get better results
194 % if the receptive fields are circular.
195 V1_circle = draw_circle( r_A_o );
196 for ( x=1:(N_d^2) )
197    mu_aff_ON_V1( :,:,x ) = mu_aff_ON_V1(:,:,x) .* V1_circle;
198    mu_aff_OFF_V1( :,:,x ) = mu_aff_OFF_V1(:,:,x) .* V1_circle;
199 end;
200 clear x;
201 LGN_circle = draw_circle( r_A );
```

```
202
203             MT_circle = draw_circle( r_MT );
204             for ( x=1:(MT_d^2) )
205                mu_aff_1_MT( :,:,x ) = mu_aff_1_MT(:,:,x) .* MT_circle;
206                mu_aff_2_MT( :,:,x ) = mu_aff_2_MT(:,:,x) .* MT_circle;
207                mu_aff_3_MT( :,:,x ) = mu_aff_3_MT(:,:,x) .* MT_circle;
208                mu_aff_4_MT( :,:,x ) = mu_aff_4_MT(:,:,x)  * MT_circle;
209             end;
210
211
212             %
213             % Initialize FIXED weights for LGN connections.
214             %
215             % num = the receptive field that it being looked at
216             %
217
218             a = 1;
219             b = 1;
220
221             sum_cent = 0;
222             sum_surr = 0;
223             num = ( (a-1)*LGN_d + b );
224
225             startx = a - ( r_A - 0.5 );
226             endx   = a + ( r_A - 0.5 );
227             starty = b - ( r_A - 0.5 );
228             endy   = b + ( r_A - 0.5 );
229
230             %
231             % Get totals for normalization.  I can't think of a better
232             % way to do this and Matlab 6.5 seems to vectorize loops
233             % pretty well.
234             %
235                          for ( x=startx:endx )
236                 for ( y=starty:endy )
237                     if ( LGN_circle( x-startx+1, y-starty+1 ) )
238                                          sum_cent = sum_cent + normgaus( a,x,b,y, sigma_c );
239                                          sum_surr = sum_surr + normgaus( a,x,b,y, sigma_s );
240                     end;
241                 end;
242                          end;
243
244             %
245             % since the afferent weights for on/off are just opposites we
246             % can get away with just computing one.  Later we will
247             % multiply this by negative one.
248             %
249             % temp_cent determines the width of the central Gaussian
```

```
% temp_surr determines the width of the surround Gaussian
%
% I believe there was an error in Bednars work here - the
% top equations top equations should be added
% ie: (x-x_o)^2 + (y-y_o)^2   not  (x-x_o)^2 - (y-y_o)^2
%
                for ( x=startx:endx )
    for ( y=starty:endy )
        if ( LGN_circle( x-startx+1, y-starty+1 ) )
                                temp_cent = normgaus( a,x,b,y, sigma_c ) / sum_cent;;;
                                temp_surr = normgaus( a,x,b,y, sigma_s ) / sum_surr;;;
                                mu_aff_LGN(  x-startx+1, y-starty+1, num ) = ...
                                        temp_cent - temp_surr;
        end;
    end;
                end;

                for ( x = 1:(LGN_d*LGN_d) )
    mu_aff_LGN( :, :, x ) = mu_aff_LGN( :, :, 1 );
                end;

                clear sum_cent sum_surr temp_cent temp_surr a b x y ...
      tenth num startx endx starty endy;
                disp( 'done - LGN Fixed weights are now initialized' );

                tenth = round( N_d^2/10 );

                E_circle = draw_circle(r_E);
                I_circle = draw_circle(r_I);



                % Initialize Gaussian starting weights for the
% excitatory/inhibitory lateral connections.

                a = 1;
                b = 1;

                sum_E = 0;
                sum_I = 0;
                num = ( (a-1)*N_d + b );

                startx = a - ( r_E - 0.5 );
                endx   = a + ( r_E - 0.5 );
                starty = b - ( r_E   0.5 );
                endy   = b + ( r_E - 0.5 );

                for ( x=startx:endx )
```

```
298                     for ( y=starty:endy )
299                                             if ( E_circle(x-startx+1,y-starty+1 ) )
300                                                 sum_E = sum_E + normgaus( x, a, y, b, sigma_E );
301                         end;
302                     end;
303                                 end;
304
305                                 for ( x=startx:endx )
306                     for ( y=starty:endy )
307                                             if ( E_circle(x-startx+1,y-starty+1 ) )
308                                                 E_V1( num, x-startx+1, y-starty+1 ) = ...
309                                         normgaus( x, a, y, b, sigma_E ) / sum_E;
310                         end;
311                     end;
312                                 end;
313
314                 for ( x=1:(N_d*N_d) )
315                     E_V1(x,:,:) = E_V1(1,:,:);
316                 end;
317
318                 startx = a - ( r_I - 0.5 );
319                 endx   = a + ( r_I - 0.5 );
320                 starty = b - ( r_I - 0.5 );
321                 endy   = b + ( r_I - 0.5 );
322
323                 for ( x=startx:endx )
324                     for ( y=starty:endy )
325                                             if ( I_circle(x-startx+1,y-starty+1 ) )
326                                                 sum_I = sum_I + normgaus( x, a, y, b, sigma_I );
327                         end;
328                     end;
329                 end;
330
331                 for ( x=startx:endx )
332                     for ( y=starty:endy )
333                                             if ( I_circle(x-startx+1,y-starty+1 ) )
334                                                 I_V1( num, x-startx+1, y-starty+1 ) = ...
335                                             normgaus( x, a, y, b, sigma_I ) / sum_I;
336                         end;
337                     end;
338                 end;
339
340                 for ( x=1:(N_d*N_d) )
341                     I_V1(x,:,:) = I_V1(1,:,:);
342                 end;
343
344
345
```

```
%
% MT E/I weights
a = 1;
                b = 1;

                sum_E = 0;
                sum_I = 0;
                num = ( (a-1)*MT_d + b );

                startx = a - ( r_E - 0.5 );
                endx   = a + ( r_E - 0.5 );
                starty = b - ( r_E - 0.5 );
                endy   = b + ( r_E - 0.5 );

                for ( x=startx:endx )
    for ( y=starty:endy )
                        if ( E_circle(x-startx+1,y-starty+1 ) )
                            sum_E = sum_E + normgaus( x, a, y, b, sigma_E ); ;
        end;
    end;
                end;

                for ( x=startx:endx )
    for ( y=starty:endy )
                        if ( E_circle(x-startx+1,y-starty+1 ) )
                            E_MT( num, x-startx+1, y-starty+1 ) =
                    normgaus( x, a, y, b, sigma_E ) / sum_E;
        end;
    end;
                end;

for ( x=1:(MT_d*MT_d) )
    E_MT(x,:,:) = E_MT(1,:,:);
end;

startx = a - ( r_I - 0.5 );
endx   = a + ( r_I   0.5 );
starty = b - ( r_I   0.5 );
endy   = b + ( r_I   0.5 );

for ( x=startx:endx )
    for ( y=starty:endy )
                        if ( I_circle(x-startx+1,y-starty+1 ) )
                            sum_I = sum_I + normgaus( x, a, y, b, sigma_I );
        end;
    end;
end;
```

```
394                 for ( x=startx:endx )
395                     for ( y=starty:endy )
396                                         if ( I_circle(x-startx+1,y-starty+1 ) )
397                                             I_MT( num, x-startx+1, y-starty+1 ) =
398                                         normgaus( x, a, y, b, sigma_I ) / sum_I;
399                         end;
400                     end;
401                 end;
402
403                 for ( x=1:(MT_d*MT_d) )
404                     I_MT(x,:,:) = I_MT(1,:,:);
405                 end;
406
407
408
409
410
411                 %
412                 % Again, I am not goind to perform face processing so ignore
413                 % these connections
414                 %E_FSA = E_V1;
415                 %I_FSA = I_V1;
416
417                 clear sum_I sum_E a b x y tenth num startx endx starty endy;
418                 disp( 'Done Initializing' );
419
420
421                 %Statistical variables
422                 testerV1 = zeros( N_d, N_d, 1 );
423                 testerMT = zeros( N_d, N_d, 1 );
424                 sum_V1 = 0;
425                 sum_MT = 0;
426
427
428                                 %
429                                 %
430                                 % Train Once
431                                 %
432                                 %
433
434
435                                 % Taken from appendix of Bednar's thesis
436                                 % gamma_N works well if you set it to 0 for gaussian (training)
437                                 % ellipses (which have only one "1" in the center of each
438                 % ellipsoid) it works well--but if you have a binary image
439                 % (e.g., solid line) then gamma_N=1 works well.
440
441                                 delta_i = 0.1;
```

```
                            beta_i = delta_i + 0.55;
                            beta = beta_i;
                            delta = delta_i;
                            alpha_A = alpha_A_i;
                            alpha_E_i = alpha_E;
                            t_si = 9;
                            t_s  = t_si;
                            gamma_N = 0.0;
                            t = 1;


        else

            %
            % We were asked to load a file instead.
            %
            load(strcat(training_pathname, training_filename));
            loadFile = 0;
            disp( 'File Loaded' )

        end;
        pack;
        startOver = 0;

    end;
    %
    % End Initialization
    %*************************************************************************




    %*************************************************************************
    % Start running training loops
    %

    % Do for t training iterations... ( 10000-20000 ideally )
    while ( training == 1 )

        % This allows for the user to press a button.  In general these
        % computations take a very long time so responses from the user
        % seem to be difficult to get.  I am not aware of a better way of
        % doing this however.
        pause( 5 );
        if( training == 0 )
            continue;
        end;
```

```
490
491
492             %increment gamma_N
493             gamma_N = t / 10000;
494             if ( gamma_N > 1 )
495                 gamma_N = 1;
496             end;
497
498
499
500             % initialize receptor, LGN reactions to zero.
501             xi_REC = zeros( R_d );
502             eta_LGN_ON = zeros( LGN_d );
503             eta_LGN_OFF = zeros( LGN_d );
504
505             if ( mod( t, 5  )==0 )
506                 disp( [ num2str( t ), ' reached' ] );
507             end;
508
509
510             % Update parameters if necessary (again, taken from the appendix).
511             switch ( t )
512                 case 0
513                     r_E = max( r_E_f, r_E_i );
514                     delta = delta_i;
515                     beta = beta_i;
516                     alpha_A = alpha_A_i;
517                     r_E = floor( r_E ) + 0.5;
518                     E_circle = draw_circle( r_E );
519                 case 200
520                     r_E = max( r_E_f, 0.6 * r_E_i );
521                     delta = delta_i + 0.01;
522                     beta = beta_i + 0.01;
523                     r_E = floor( r_E ) + 0.5;
524                     E_circle = draw_circle( r_E );
525                 case 500
526                     r_E = max( r_E_f, 0.42 * r_E_i );
527                     delta = delta_i + 0.02;
528                     beta = beta_i + 0.02;
529                     alpha_A = 50/70 * alpha_A_i;
530                     alpha_E = 0.5 * alpha_E_i;
531                     r_E = floor( r_E ) + 0.5;
532                     E_circle = draw_circle( r_E );
533                 case 1000
534                     r_E = max( r_E_f, 0.336*r_E_i );
535                     delta = delta_i + 0.05;
536                     beta = beta_i + 0.02;
537                     r_E = floor( r_E ) + 0.5;
```

```
538                 E_circle = draw_circle( r_E );
539             case 2000
540                 r_E = max( r_E_f, 0.269*r_E_i );
541                 delta = delta_i + 0.08;
542                 beta = beta_i + 0.05;
543                 t_s = t_s + 1;
544                 r_E = floor( r_E ) + 0.5;
545                 alpha_A = 40/70 * alpha_A_i;
546                 E_circle = draw_circle( r_E );
547             case 3000
548                 r_E = max( r_E_f, 0.215*r_E_i );
549                 delta = delta_i + 0.10;
550                 beta = beta_i + 0.08;
551                 r_E = floor( r_E ) + 0.5;
552                 E_circle = draw_circle( r_E );
553             case 4000
554                 r_E = max( r_E_f, 0.129*r_E_i );
555                 delta = delta_i + 0.10;
556                 beta = beta_i + 0.11;
557                 r_E = floor( r_E ) + 0.5;
558                 alpha_A = alpha_A_i * 30/70;
559                 E_circle = draw_circle( r_E );
560             case 5000
561                 r_E = max( r_E_f, 0.077*r_E_i );
562                 delta = delta_i + 0.11;
563                 beta = beta_i + 0.14;
564                 t_s = t_s + 1;
565                 r_E = floor( r_E ) + 0.5;
566                 E_circle = draw_circle( r_E );
567             case 6500
568                 r_E = max( r_E_f, 0.046*r_E_i );
569                 delta = delta_i + 0.12;
570                 beta = beta_i + 0.17;
571                 t_s = t_s + 1;
572                 r_E = floor( r_E ) + 0.5;
573                 E_circle = draw_circle( r_E );
574             case 8000
575                 r_E = max( r_E_f, 0.028*r_E_i );
576                 delta = delta_i + 0.13;
577                 beta = beta_i + 0.20;
578                 r_E = floor( r_E ) + 0.5;
579                 E_circle = draw_circle( r_E );
580         end;
581
582
583         % If necessary, trim V1-r_E afferent weights.
584         old_r_E = (sum( size( E_V1( 1, 1, : ) ) )-2) / 2;
585         while ( old_r_E > r_E )
```

```
586                 E_V1 = E_V1( :, 2:2*old_r_E-1, 2:2*old_r_E-1 );
587                 old_r_E = (sum( size( E_V1( 1, 1, : ) ) )-2) / 2;
588             end;
589
590             clear old_r_E;
591
592
593             %
594             % For the first 100 iterations the system needs to be pretrained.
595             % Therefore we create random elipsoids and feed these into the
596             % system.
597             %
598
599             if( t < trainingIterations )
600                 % Create ellipsoids ( activation for LGN ).
601                 for ( i = 1:4 )
602                     x_o = round( rand(1) * N_d + r_A_o );
603                     y_o = round( rand(1) * N_d + r_A_o );
604                     %theta = deg2rad( rand(1) * 180 );
605                     theta = rand(1) * pi;
606                     for ( x = 1:R_d )
607                         for ( y = 1:R_d )
608                                                 temp = -1 * (((x-x_o) * cos(theta) - (y-y_o) * ...
609                                                                 sin(theta))^2 / A^2);
610                                                 temp = temp - (((x-x_o) * sin(theta) + ...
611                                                      (y-y_o) * cos(theta))^2 / B^2);
612                                                 temp = exp( temp );
613                                                 xi_REC( x, y ) = xi_REC( x, y ) + temp;
614                                             end;
615                     end;
616                 end;
617                 clear x_o y_o theta x y i temp;
618             else
619                 if( t < (controlIterations + trainingIterations ) )
620
621                     if( controlSet == 1 )
622                         xi_REC = imresize( double( imread( ...
623                                     strcat( 'images/images1/', ...
624                                     num2str( this_rounds_image ) , ...
625                                     '.bmp' ) ) ) / 255, [R_d R_d] );
626                         this_rounds_image = this_rounds_image + 1;
627                         if( this_rounds_image > 18 )
628                             this_rounds_image = 1;
629                         end
630                     elseif( controlSet == 2 )
631                         xi_REC = imresize( double( imread( strcat( ...
632                                     'images/images1/', num2str( ...
633                                     this_rounds_image ) , '.bmp' ) ) ) ...
```

```
                    / 255, [R_d R_d] );
    this_rounds_image = this_rounds_image   1;
    if( this_rounds_image < 0 )
        this_rounds_image = 18;
    end
elseif( controlSet == 3 )
    mov = aviread( 'images/newShortCars.AVI', 74 + ...
                                        this_rounds_image );
            whole_image = zeros( R_d, R_d, size( mov, 2 ) );

                for( i=1:size( mov, 2 ) )
                    whole_image(:,:,i) = rgb2gray( imresize( ...
                        mov( 1, i ).cdata, [R_d R_d] ) );
                end;
                whole_image = whole_image ./ 255;
                xi_REC = whole_image( :, :, 1 );

    this_rounds_image = this_rounds_image + 1;
    if( this_rounds_image > 100 )
        this_round_image = 1;
    end;
elseif( controlSet == 4 )
    mov = aviread( 'images/newShortCars.AVI', 174 + ...
                                        this_rounds_image );
            whole_image = zeros( R_d, R_d, size( mov, 2 ) );

                for( i=1:size( mov, 2 ) )
                    whole_image(:,:,i) = rgb2gray( imresize( ...
                           mov( 1, i ).cdata, [R_d R_d] ) );
                end;
                whole_image = whole_image ./ 255;
                xi_REC = whole_image( :, :, 1 );

    this_rounds_image = this_rounds_image + 1;
    if( this_rounds_image > 100 )
        this_round_image = 1;
    end;
elseif( controlSet == 5 )
    xi_REC = imresize( double( imread( ...
            'images/horizontal.bmp' ) ) / 255, [R_d R_d] );
elseif( controlSet == 6 )
    xi_REC = imresize( double( imread( ...
             'images/vertical.bmp' ) ) / 255, [R_d R_d] );
elseif( controlSet == 7 )
    xi_REC = imresize( double( imread( 'images/circle.bmp' ) ) / 255, [R_d R_d]
else
    xi_REC = imresize( double( imread( strcat( ...
                'images/images', num2str( controlSet ), ...
```

```
682                             '/', num2str( this_rounds_image ) , ...
683                             '.bmp' ) ) ) / 255, [R_d R_d] );
684                     this_rounds_image = this_rounds_image + 1;
685                     if( this_rounds_image > 18 )
686                         this_rounds_image = 1;
687                     end
688                 end
689
690             else
691
692                 if( t == ( controlIterations + trainingIterations ) )
693                     this_rounds_image = 1;
694                 end;
695
696                 if( expSet == 1 )
697                     xi_REC = imresize( double( imread( strcat( ...
698                             'images/images1/', num2str( ...
699                             this_rounds_image ) , '.bmp' ) ) ) / ...
700                             255, [R_d R_d] );
701                     this_rounds_image = this_rounds_image + 1;
702                     if( this_rounds_image > 18 )
703                         this_rounds_image = 1;
704                     end
705                 elseif( expSet == 2 )
706                     xi_REC = imresize( double( imread( strcat( ...
707                             'images/images1/', num2str( ...
708                             this_rounds_image ) , '.bmp' ) ) ) / ...
709                             255, [R_d R_d] );
710                     this_rounds_image = this_rounds_image   1;
711                     if( this_rounds_image < 0 )
712                         this_rounds_image = 18;
713                     end
714                 elseif( expSet == 3 )
715                     mov = aviread( 'images/newShortCars.AVI', 74 + ...
716                                 this_rounds_image );
717                         whole_image = zeros( R_d, R_d, size( mov, 2 ) );
718
719                             for( i=1:size( mov, 2 ) )
720                                 whole_image(:,:,i) = rgb2gray( imresize( ...
721                                     mov( 1, i ).cdata, [R_d R_d] ) );
722                             end;
723                             whole_image = whole_image ./ 255;
724                             xi_REC = whole_image( :, :, 1 );
725
726                     this_rounds_image = this_rounds_image + 1;
727                     if( this_rounds_image > 100 )
728                         this_round_image = 1;
729                     end;
```

```
730                 elseif( expSet == 4 )
731                     mov = aviread( 'images/newShortCars.AVI', 174 + ...
732                                                         this_rounds_image );
733                             whole_image = zeros( R_d, R_d, size( mov, 2 ) );
734 
735                                 for( i=1:size( mov, 2 ) )
736                                     whole_image(:,:,i) = rgb2gray( imresize( ...
737                                         mov( 1, i ).cdata, [R_d R_d] ) );
738                                 end;
739                                 whole_image = whole_image ./ 255;
740                                 xi_REC = whole_image( :, :, 1 );
741 
742                     this_rounds_image = this_rounds_image + 1;
743                     if( this_rounds_image > 100 )
744                         this_round_image = 1;
745                     end;
746                 elseif( expSet == 5 )
747                     xi_REC = imresize( double( imread( ...
748                             'images/horizontal.bmp' ) ) / 255, [R_d R_d] );
749                 elseif( expSet == 6 )
750                     xi_REC = imresize( double( imread( ...
751                             'images/vertical.bmp' ) ) / 255, [R_d R_d] );
752                 elseif( expSet == 7 )
753                     if( mod( this_rounds_image, 2 ) == 0 )
754                         xi_REC = imresize( double( imread( ...
755                             'images/vertical.bmp' ) ) / 255, [R_d R_d] );
756                     else
757                         xi_REC = imresize( double( imread( ...
758                             'images/horizontal.bmp' ) ) / 255, [R_d R_d] );
759                     end;
760                     this_rounds_image = this_rounds_image + 1;
761                 else
762                     xi_REC = imresize( double( imread( strcat( ...
763                             'images/images', num2str( expSet ), '/', ...
764                             num2str( this_rounds_image ) , '.bmp' ...
765                             ) ) ) / 255, [R_d R_d] );
766                     this_rounds_image = this_rounds_image + 1;
767                     if( this_rounds_image > 18 )
768                         this_rounds_image = 1;
769                     end;
770                 end;
771 
772             end;
773         end;
774 
775 
776         % Debug purposes only.
777         % xi_REC = image;
```

```
778
779             % Threshhold to 1 or below.
780             xi_REC  = (xi_REC > 1 ) + ((xi_REC  <= 1) .* xi_REC );
781
782
783
784             % start calculating xi and eta for LGN
785             xi_LGN_ON = zeros( LGN_d );
786             xi_LGN_OFF = zeros( LGN_d );
787
788             for ( a = 1:LGN_d )
789                 for ( b = 1:LGN_d )
790
791                     num = (a-1)*LGN_d+b;
792                     startx = a;
793                     endx   = a + ( r_A - 0.5 )*2;
794                     starty = b;
795                     endy   = b + ( r_A - 0.5 )*2;
796
797
798                     for ( x = startx:endx )
799                         for ( y = starty:endy )
800                             if ( LGN_circle(x-startx+1,y-starty+1 ) )
801                                 xi_LGN_ON(  a, b ) = xi_LGN_ON( a, b )  + ...
802                                                      xi_REC( x, y ) * (-1 * ...
803                                                      mu_aff_LGN(  x-startx+1 ...
804                                                      , y-starty+1, num ));
805                                 xi_LGN_OFF( a, b ) = xi_LGN_OFF( a, b ) + ...
806                                                      xi_REC( x, y ) * ...
807                                                      mu_aff_LGN( x-startx+1 ...
808                                                      , y-starty+1, num );
809                             end;
810                         end;
811                     end;
812
813                 end;
814             end;
815
816             clear a b x y nm startx endx starty endy;
817             pack;
818
819             div = max(max(xi_LGN_ON));
820             if( div == 0 )
821                 div = 1;
822             end;
823
824             gamma_A = 1 / div;
825
```

```
826
827         %Sigma Function - this is repeated so that Matlab 6.5 can better
828         %optimize the code are runtime.
829         x = gamma_A .* xi_LGN_ON;
830         temp_size = size( x );
831         temp_delta = zeros( temp_size(1) ) + delta;
832         temp_beta = zeros( temp_size(1) ) + beta;
833         clear temp_size;
834
835         eta_LGN_ON = ( x > temp_delta );
836         eta_LGN_ON = eta_LGN_ON .* ( x < temp_beta );
837         clear temp_delta;
838         clear temp_beta;
839
840         eta_LGN_ON = eta_LGN_ON .* ( ( x - delta ) / ( beta - delta ) );
841         eta_LGN_ON = eta_LGN_ON + ( x >= beta );
842
843
844         div = max(max(xi_LGN_OFF));
845         if( div == 0 )
846             div = 1;
847         end;
848
849         gamma_A = 1 / div;
850
851
852
853         %Sigma Function
854         x = gamma_A .* xi_LGN_OFF;
855         temp_size = size( x );
856         temp_delta = zeros( temp_size(1) ) + delta;
857         temp_beta = zeros( temp_size(1) ) + beta;
858         clear temp_size;
859
860         eta_LGN_OFF = ( x > temp_delta );
861         eta_LGN_OFF = eta_LGN_OFF .* ( x < temp_beta );
862         clear temp_delta;
863         clear temp_beta;
864
865         eta_LGN_OFF = eta_LGN_OFF .* ( ( x - delta ) / ( beta   delta ) );
866         eta_LGN_OFF = eta_LGN_OFF + ( x >= beta );
867         % end calculating eta (reaction) for both LGN channels
868
869
870
871
872         %
873         % This is where LGN moves to V1
```

```
874             %
875
876             zeta_V1 = zeros( N_d );
877
878             % start calculating zeta (activation) for V1
879             for ( i=1:N_d )
880                 for ( j=1:N_d )
881
882                     num = (i-1)*N_d+j;
883
884                     starta = i;
885                     enda   = i + ( r_A_o - 0.5 ) * 2;
886                     startb = j;
887                     endb   = j + ( r_A_o   0.5 ) * 2;
888                     sum_zeta_V1 = 0;
889
890                     for ( a = starta:enda )
891                         for ( b = startb:endb )
892                             if ( V1_circle(a-starta+1,b-startb+1 ) )
893                                %
894                                % The original thought was to take the strongest
895                                % response and feed it to V1.  However,
896                                % experiments have shown that this is not
897                                % accurate.
898                                %
899                                %   if ( xi_LGN_ON( a, b ) * mu_aff_ON_V1( a-starta+1, b-startb+1 ) > xi
900                                %       zeta_V1( i, j ) = zeta_V1( i, j ) + xi_LGN_ON( a, b ) * mu_aff_O
901                                %       sum_zeta_V1 = sum_zeta_V1 + xi_LGN_ON( a, b );
902                                %   else
903                                %       zeta_V1( i, j ) = zeta_V1( i, j ) + xi_LGN_OFF( a, b ) * mu_aff_
904                                %       sum_zeta_V1 = sum_zeta_V1 + xi_LGN_OFF( a, b );
905                                %   end;
906                                %
907                                % Instead of taking the larger of the two lets
908                                % try and sum everything up.  This seems to
909                                % work better - but the actual mechanism for
910                                % this is still unknow.  Therefore more work
911                                % could be done to find the optimal way of
912                                % combining the on/off channels when feeding
913                                % them into V1
914                                %
915                                 zeta_V1( i, j ) = zeta_V1( i, j ) + ...
916                                                   eta_LGN_ON( a, b ) * ...
917                                                   mu_aff_ON_V1( a-starta+1, ...
918                                                   b-startb+1, num );
919                                 sum_zeta_V1 = sum_zeta_V1 + ...
920                                               eta_LGN_ON( a, b ) + ...
921                                               eta_LGN_OFF( a, b );
```

```
                        zeta_V1( i, j ) = zeta_V1( i, j ) + ...
                                          eta_LGN_OFF( a, b ) * ...
                                          mu_aff_OFF_V1( a-starta+1, ...
                                          b-startb+1, num );
                    end;
                end;
            end;

            zeta_V1( i, j ) = ( gamma_A .* zeta_V1( i, j ) ) / ...
                              ( 1 + gamma_N * sum_zeta_V1 );

        end;
    end;

    % Set the maximum to 1

    div = max(max(zeta_V1));
    if( div == 0 )
        div = 1;
    end;

    gamma_A = 1/div;
    zeta_V1 = zeta_V1 .* gamma_A;
    clear num centera centerb starta enda startb endb sum_zeta_V1 ...
          i j a b;

    % end calculating zeta for V1.

    clear i j k l num startk endk startl endl;




    % Start settling iterations. (eta V1)
    if( t > 1 )
        last_eta_V1 = eta_V1;
    end;

    for ( blah = 0:t_s );

    % Calculate eta (reaction) for V1.  If this is the first settling
    % iteration this is basically the sigma function implemented on V1.
    % Else, the reaction is "tweaked" based on the weights.

    if ( blah==0 )
        %eta_V1 = sigma_function( zeta_V1, delta, beta );

        %Sigma Function
```

```
970                 x = zeta_V1;
971                 temp_size = size( x );
972                 temp_delta = zeros( temp_size(1) ) + delta;
973                 temp_beta = zeros( temp_size(1) ) + beta;
974                 clear temp_size;
975
976                 eta_V1 = ( x > temp_delta );
977                 eta_V1 = eta_V1 .* ( x < temp_beta );
978                 clear temp_delta;
979                 clear temp_beta;
980
981                 eta_V1 = eta_V1 .* ( ( x   delta ) / ( beta   delta ) );
982                 eta_V1 = eta_V1 + ( x >= beta );
983
984                 if( t == 1 )
985                     last_eta_V1 = eta_V1;
986                 end;
987
988             else
989                 old_eta_V1 = eta_V1;
990                 for ( i=1:N_d )
991                     for ( j=1:N_d )
992
993                         num = (i-1)*N_d+j;
994
995                         sum_E = 0;
996                         sum_I = 0;
997
998                         startk = max( 1,   i   ( r_E-0.5 ) );
999                         endk   = min( N_d, i + ( r_E-0.5 ) );
1000                        startl = max( 1,   j   ( r_E-0.5 ) );
1001                        endl   = min( N_d, j + ( r_E-0.5 ) );
1002
1003                        for ( k = startk:endk )
1004                            for ( l = startl:endl )
1005                                if ( E_circle(k-startk+1,l-startl+1 ) )
1006                                    sum_E = sum_E + E_V1( num, k-startk+1, ...
1007                                            l-startl+1 ) * old_eta_V1( k, l );
1008                                end;
1009                            end;
1010                        end;
1011
1012                        startk = max( 1,   i - ( r_I-0.5 ) );
1013                        endk   = min( N_d, i + ( r_I-0.5 ) );
1014                        startl = max( 1,   j   ( r_I-0.5 ) );
1015                        endl   = min( N_d, j + ( r_I-0.5 ) );
1016
1017                        for ( k = startk:endk )
```

```
                    for ( l = startl:endl )
                        if ( I_circle(k-startk+1,l-startl+1 ) )
                            sum_I = sum_I + I_V1( num, k-startk+1, ...
                                    l-startl+1 ) * old_eta_V1( k, l );
                        end;
                    end;
                end;

                eta_V1( i, j ) = zeta_V1( i, j ) + gamma_E * ...
                                 sum_E   gamma_I * sum_I;
            end;
        end;

        clear i j k l num sum_E sum_I startk endk startl endl;
        %eta_V1 = sigma_function( eta_V1, delta, beta );


        %Sigma Function
        x = eta_V1;
        temp_size = size( x );
        temp_delta = zeros( temp_size(1) ) + delta;
        temp_beta = zeros( temp_size(1) ) + beta;
        clear temp_size;

        eta_V1 = ( x > temp_delta );
        eta_V1 = eta_V1  * ( x < temp_beta );
        clear temp_delta;
        clear temp_beta;

        eta_V1 = eta_V1 .* ( ( x - delta ) / ( beta - delta ) );
        eta_V1 = eta_V1 + ( x >= beta );

    end;
    % end calculating eta V1 (reaction).



    % Now update weights based on V1 reactions.  (This is the self
    % organization!!!)  IT seems to work well   but I am curious to
    % see if there are better ways to update these weights - or
    % perhaps faster ways by looking at approximations of regions.
    % If this is the case then it might be possible to distribute a
    % lot of these computations and would drastically speed things up.
    old_mu_aff_ON_V1 = mu_aff_ON_V1;
    old_mu_aff_OFF_V1 = mu_aff_OFF_V1;
    old_E_V1 = E_V1;
    old_I_V1 = I_V1;
```

```
% Update V1 afferent weights.
for ( i=1:N_d )
    for ( j = 1:N_d )

        num = ( i-1 ) * N_d + j;
        sum_mu_aff_ON_V1 = 0;
        sum_mu_aff_OFF_V1 = 0;

        startm = i;
        endm   = i + ( r_A_o   0.5 ) * 2;
        startn = j;
        endn   = j + ( r_A_o   0.5 ) * 2;

        for ( m=startm:endm )
            for ( n=startn:endn )
                if ( V1_circle(m-startm+1,n-startn+1 ) )
                    sum_mu_aff_ON_V1 = sum_mu_aff_ON_V1 + ...
                                       old_mu_aff_ON_V1( ...
                                       m-startm+1, n-startn+1, ...
                                       num ) + alpha_A * ...
                                       eta_V1( i, j ) * ...
                                       eta_LGN_ON( m, n );
                    sum_mu_aff_OFF_V1 = sum_mu_aff_OFF_V1 + ...
                                        old_mu_aff_OFF_V1( ...
                                        m-startm+1, n-startn+1, ...
                                        num ) + alpha_A * ...
                                        eta_V1( i, j ) * ...
                                        eta_LGN_OFF( m, n );
                end;
            end;
        end;

        for ( m=startm:endm )
            for ( n = startn:endn )
                if ( V1_circle(m-startm+1,n-startn+1 ) )
                    mu_aff_ON_V1( m-startm+1, n-startn+1, num ) ...
                        = ( old_mu_aff_ON_V1( m-startm+1, ...
                            n-startn+1, num ) + alpha_A * ...
                            eta_V1( i, j ) * ...
                            eta_LGN_ON( m, n ) ) / ...
                            sum_mu_aff_ON_V1;
                    mu_aff_OFF_V1( m-startm+1, n-startn+1, num ) ...
                        = ( old_mu_aff_OFF_V1( m-startm+1, ...
                            n-startn+1, num ) + alpha_A * ..
                            eta_V1( i, j ) * ...
                            eta_LGN_OFF( m, n ) ) / ...
                            sum_mu_aff_OFF_V1;
                end;
```

```
                end;
            end;
        end;
    end;

    clear i j num sum_mu_aff_ON_V1 sum_mu_aff_OFF_V1 startm ...
          endm startn endn old_mu_aff_ON_V1 old_mu_aff_OFF_V1;



    % Update V1 lateral weights.
    for ( i = 1:N_d )
        for ( j = 1:N_d )

            sum_E_V1 = 0;
            sum_I_V1 = 0;
            num = (i-1)*N_d+j;

            startm = max( 1,   i   (r_E-0.5) );
            endm   = min( N_d, i + (r_E-0.5) );
            startn = max( 1,   j   (r_E-0.5) );
            endn   = min( N_d, j + (r_E-0.5) );

            for ( m=startm:endm )
                for ( n = startn:endn )
                        if ( E_circle(m-startm+1,n-startn+1 ) )
                            sum_E_V1 = sum_E_V1 + old_E_V1( num, ...
                                       m-startm+1, n-startn+1 ) + ...
                                       alpha_E * eta_V1( i, j ) * ...
                                       eta_V1( m, n );
                        end;
                end;
            end;

           for ( m = startm:endm )
                for ( n = startn:endn )
                    if ( E_circle(m-startm+1,n-startn+1 ) )
                        E_V1( num, m-startm+1, n-startn+1 ) = ...
                                (old_E_V1( num, m-startm+1, ...
                                n-startn+1 ) + alpha_E * ...
                                eta_V1( i, j ) * eta_V1( m, n )) ...
                                / sum_E_V1;
                    else
                        E_V1(num,m-startm+1,n-startn+1 ) = 0;
                    end;
                end;
            end;
```

```
1162                    startm = max( 1, i - (r_I-0.5) );
1163                    endm   = min( N_d, i + (r_I-0.5) );
1164                    startn = max( 1, j - (r_I-0.5) );
1165                    endn   = min( N_d, j + (r_I-0.5) );
1166
1167                    for ( m=startm:endm )
1168                        for ( n = startn:endn )
1169                            if ( I_circle(m-startm+1,n-startn+1 ) )
1170                               sum_I_V1 = sum_I_V1 + old_I_V1( num, ...
1171                                          m-startm+1, n-startn+1 ) + ...
1172                                          alpha_I * eta_V1( i, j ) * ...
1173                                          eta_V1( m, n );
1174                               end;
1175                        end;
1176                    end;
1177
1178
1179
1180                    for ( m = startm:endm )
1181                        for ( n = startn:endn )
1182                            if ( I_circle(m-startm+1,n-startn+1 ) )
1183                               I_V1( num, m-startm+1, n-startn+1 ) = ...
1184                                    (old_I_V1( num, m-startm+1, ...
1185                                    n-startn+1 ) + alpha_I * ...
1186                                    eta_V1( i, j ) * eta_V1( m, n )) / sum_I_V1;
1187                               end;
1188                        end;
1189                    end;
1190
1191
1192                end;
1193            end;
1194
1195            % Finish updating lateral weights (and all weights).
1196
1197            clear i j m n num old_E_V1 startm endm startn endn ...
1198                  sum_I_V1 sum_E_V1;
1199            end; % End settling iterations.
1200
1201            clear blah;
1202
1203
1204
1205
1206
1207            %
1208            % This will  test for inconsistancies between images
1209            % Images should be continuous for this to be meaningfull because it
```

```
% looks at the response between two iterations in order to cancel of
% the effect of "seeing something out of the corner of your eye".
% In other words, in human vision when we first see things our
% attention is quickly focused   not because we don't understand
% what we are seeing but because something drastic in the image
% has changed.
%
% What remains is what I call the unknown response.  It is what the
% system must learn about the system.
%
unknown_response = abs( eta_V1 - last_eta_V1 ) > 0;
temp = (1-(eta_LGN_ON - eta_LGN_OFF));
size1 = size( eta_LGN_ON );
size2 = size( unknown_response );
s1 = size1(1) - size2(1);
s2 = s1 / 2;

unknown_response = unknown_response .* eta_V1;
clear temp;

%
% Now we must look at processing motion.  For this we continue the
% above process in the same fasion feeding information to two
% different areas, MT and MST.  Later, this information will need
% to be feedback into themselves and/or the V1 layer.  In order
% to keep orrientation processing my current feeling is that the
% V1 layer should stay as it is and that MT and MST should be
% designed to feed into one another.
%

zeta_MT = zeros( MT_d );

if( MT_NOT_SET )
    %eta_V1_5 = eta_V1;
    %eta_V1_4 = eta_V1;
    %eta_V1_3 = eta_V1;
    %eta_V1_2 = eta_V1;
    %eta_V1_1 = eta_V1;
    eta_V1_5 = unknown_response;
    eta_V1_4 = unknown_response;
    eta_V1_3 = unknown_response;
    eta_V1_2 = unknown_response;
    eta_V1_1 = unknown_response;

end


div = abs(min(min( unknown_response )));
```

```
1258            if( div == 0 )
1259                div = abs( max( max( unknown_response )));
1260            end;
1261
1262            if( div == 0 )
1263                div = 1;
1264            end;
1265
1266            eta_V1_5 = eta_V1_4;
1267            eta_V1_4 = eta_V1_3;
1268            eta_V1_3 = eta_V1_2;
1269            eta_V1_2 = eta_V1_1;
1270            %eta_V1_1 = eta_V1;
1271            eta_V1_1 = unknown_response;
1272
1273
1274            % start calculating zeta (activation) for MT
1275            for ( i=1:MT_d )
1276                for ( j=1:MT_d )
1277
1278                    num = (i-1)*MT_d+j;
1279
1280                    starta = i;
1281                    enda   = i + ( r_MT - 0.5 ) * 2;
1282                    startb = j;
1283                    endb   = j + ( r_MT - 0.5 ) * 2;
1284
1285                    if( enda > N_d ) enda = N_d; end
1286                    if( endb > N_d ) endb = N_d; end
1287
1288                    sum_zeta_MT = 0;
1289
1290                    for ( a = starta:enda )
1291                        for ( b = startb:endb )
1292                            if ( MT_circle(a-starta+1,b-startb+1 ) )
1293
1294                               %
1295                               % As a first attempt I am simply going to combine
1296                               % all of the previous layers responeses.
1297                               % Therefore more work could be done to find
1298                               % the optimal way of combining the on/off
1299                               % channels when feeding them into MT
1300                               %
1301
1302                                zeta_MT( i, j ) = zeta_MT( i, j ) + ...
1303                                                  eta_V1_1( a, b ) * ...
1304                                                  mu_aff_1_MT( a-starta+1, ...
1305                                                  b-startb+1, num ) + ...
```

```
                                        eta_V1_2( a, b ) * ...
                                        mu_aff_2_MT( a-starta+1, ...
                                        b-startb+1, num ) + ...
                                        eta_V1_3( a, b ) * ...
                                        mu_aff_3_MT( a-starta+1, ...
                                        b-startb+1, num ) + ...
                                        eta_V1_4( a, b ) * ...
                                        mu_aff_4_MT( a-starta+1, ...
                                        b-startb+1, num );
                        sum_zeta_MT = sum_zeta_MT + ...
                                      eta_V1_1( a, b ) + ...
                                      eta_V1_2( a, b ) + ...
                                      eta_V1_3( a, b ) + ...
                                      eta_V1_4( a, b );
                    end;
                end;
            end;

            zeta_MT( i, j ) = ( gamma_A_MT .* zeta_MT( i, j ) ) / ...
                              ( 1 + gamma_N_MT * sum_zeta_MT );

        end;
    end;

    % Set the maximum to 1
    div = max(max(sum_zeta_MT));
    %div = max(max(xi_V1));
    if( div == 0 )
        div = 1;
    end;

    gamma_A_MT = 1/div;
    zeta_MT = zeta_MT .* gamma_A_MT;
    clear num centera centerb starta enda startb endb sum_zeta_MT ...
          i j a b;
    % end calculating zeta for MT.

    clear i j k l num startk endk startl endl;



    % Start settling iterations. (eta MT)
    for ( blah = 0:t_s );

    % Calculate eta (reaction) for MT.  If this is the first settling
    % iteration this is basically the sigma function implemented on MT.
    % Else, the reaction is "tweaked" based on the weights.
```

```
1354            if ( blah==0 )
1355                %eta_MT = sigma_function( zeta_MT, delta, beta );
1356
1357                %Sigma Function
1358                x = zeta_MT;
1359                temp_size = size( x );
1360                temp_delta = zeros( temp_size(1) ) + delta;
1361                temp_beta = zeros( temp_size(1) ) + beta;
1362                clear temp_size;
1363
1364                eta_MT = ( x > temp_delta );
1365                eta_MT = eta_MT .* ( x < temp_beta );
1366                clear temp_delta;
1367                clear temp_beta;
1368
1369                eta_MT = eta_MT .* ( ( x - delta ) / ( beta   delta ) );
1370                eta_MT = eta_MT + ( x >= beta );
1371
1372            else
1373                old_eta_MT = eta_MT;
1374                for ( i=1:MT_d )
1375                    for ( j=1:MT_d )
1376
1377                        num = (i-1)*MT_d+j;
1378
1379                        sum_E = 0;
1380                        sum_I = 0;
1381
1382                        startk = max( 1,   i - ( r_E-0.5 ) );
1383                        endk   = min( MT_d, i + ( r_E-0.5 ) );
1384                        startl = max( 1,   j - ( r_E-0.5 ) );
1385                        endl   = min( MT_d, j + ( r_E-0.5 ) );
1386
1387                        for ( k = startk:endk )
1388                            for ( l = startl:endl )
1389                                if ( E_circle(k-startk+1,l-startl+1 ) )
1390                                    sum_E = sum_E + E_MT( num, k-startk+1, ...
1391                                            l-startl+1 ) * old_eta_MT( k, l );
1392                                end;
1393                            end;
1394                        end;
1395
1396                        startk = max( 1,   i - ( r_I-0.5 ) );
1397                        endk   = min( MT_d, i + ( r_I-0.5 ) );
1398                        startl = max( 1,   j - ( r_I-0.5 ) );
1399                        endl   = min( MT_d, j + ( r_I-0.5 ) );
1400
1401                        for ( k = startk:endk )
```

```
                    for ( l = startl:endl )
                        if ( I_circle(k-startk+1,l-startl+1 ) )
                            sum_I = sum_I + I_MT( num, k-startk+1, ...
                                    l-startl+1 ) * old_eta_MT( k, l );
                        end;
                    end;
                end;

                eta_MT( i, j ) = zeta_MT( i, j ) + gamma_E * ...
                                 sum_E - gamma_I * sum_I;
            end;
        end;

        clear i j k l num sum_E sum_I startk endk startl endl;
        %eta_MT = sigma_function( eta_MT, delta, beta );


        %Sigma Function
        x = eta_MT;
        temp_size = size( x );
        temp_delta = zeros( temp_size(1) ) + delta;
        temp_beta = zeros( temp_size(1) ) + beta;
        clear temp_size;

        eta_MT = ( x > temp_delta );
        eta_MT = eta_MT .* ( x < temp_beta );
        clear temp_delta;
        clear temp_beta;

        eta_MT   eta_MT .* ( ( x - delta ) / ( beta   delta ) );
        eta_MT   eta_MT + ( x >= beta );

    end;
    % end calculating eta MT (reaction).


    if( t > 1 )
        eta_MT_4 = eta_MT_3;
        eta_MT_3 = eta_MT_2;
        eta_MT_2 = eta_MT_1;
        eta_MT_1 = eta_MT;
    else
        eta_MT_4 = eta_MT;
        eta_MT_3 = eta_MT;
        eta_MT_2 = eta_MT;
        eta_MT_1 = eta_MT;
    end
```

```
1450
1451        % Now update weights based on MT reactions.  (This is the self
1452        % organization!!!)  IT seems to work well - but I am curious to
1453        % see if there are better ways to update these weights - or
1454        % perhaps faster ways by looking at approximations of regions.
1455        % If this is the case then it might be possible to distribute a
1456        % lot of these computations and would drastically speed things up.
1457
1458        old_mu_aff_1_MT = mu_aff_1_MT;
1459        old_mu_aff_2_MT = mu_aff_2_MT;
1460        old_mu_aff_3_MT = mu_aff_3_MT;
1461        old_mu_aff_4_MT = mu_aff_4_MT;
1462        old_E_MT = E_MT;
1463        old_I_MT = I_MT;
1464
1465        for ( i=1:MT_d )
1466            for ( j = 1:MT_d )
1467
1468                num = ( i-1 ) * MT_d + j;
1469                sum_mu_aff_1_MT = 0;
1470                sum_mu_aff_2_MT = 0;
1471                sum_mu_aff_3_MT = 0;
1472                sum_mu_aff_4_MT = 0;
1473
1474                startm = i;
1475                endm   = i + ( r_MT   0.5 ) * 2;
1476                startn = j;
1477                endn   = j + ( r_MT - 0.5 ) * 2;
1478
1479                for ( m=startm:endm )
1480                    for ( n=startn:endn )
1481                        if ( MT_circle(m-startm+1,n-startn+1 ) )
1482
1483                            sum_mu_aff_1_MT = sum_mu_aff_1_MT + ...
1484                                old_mu_aff_1_MT( m-startm+1, ...
1485                                n-startn+1, num ) + alpha_A * ...
1486                                eta_MT_1( i, j ) * eta_V1( m, n );
1487                            sum_mu_aff_2_MT = sum_mu_aff_2_MT + ...
1488                                old_mu_aff_2_MT( m-startm+1, ...
1489                                n-startn+1, num ) + alpha_A * ...
1490                                eta_MT_2( i, j ) * eta_V1( m, n );
1491                            sum_mu_aff_3_MT = sum_mu_aff_3_MT + ...
1492                                old_mu_aff_3_MT( m-startm+1, ...
1493                                n-startn+1, num ) + alpha_A * ...
1494                                eta_MT_3( i, j ) * eta_V1( m, n );
1495                            sum_mu_aff_4_MT = sum_mu_aff_4_MT + ...
1496                                old_mu_aff_4_MT( m-startm+1, ...
1497                                n-startn+1, num ) + alpha_A * ...
```

```
                                eta_MT_4( i, j ) * eta_V1( m, n );

                        end;
                    end;
                end;

                for ( m=startm:endm )
                    for ( n = startn:endn )
                        if ( MT_circle(m-startm+1,n-startn+1 ) )

                            mu_aff_1_MT( m-startm+1, n-startn+1, num ) ...
                                = ( old_mu_aff_1_MT( m-startm+1, ...
                                    n-startn+1, num ) + alpha_A * ...
                                    eta_MT_1( i, j ) * eta_V1( m, n ) ) ...
                                    / sum_mu_aff_1_MT;
                            mu_aff_2_MT( m-startm+1, n-startn+1, num )
                                = ( old_mu_aff_2_MT( m-startm+1, ...
                                    n-startn+1, num ) + alpha_A * ...
                                    eta_MT_2( i, j ) * eta_V1( m, n ) ) ...
                                    / sum_mu_aff_2_MT;
                            mu_aff_3_MT( m-startm+1, n-startn+1, num ) ...
                                = ( old_mu_aff_3_MT( m-startm+1, ...
                                    n-startn+1, num ) + alpha_A * ...
                                    eta_MT_3( i, j ) * eta_V1( m, n ) ) ...
                                    / sum_mu_aff_3_MT;
                            mu_aff_4_MT( m-startm+1, n-startn+1, num ) ...
                                = ( old_mu_aff_4_MT( m-startm+1, ...
                                    n-startn+1, num ) + alpha_A * ...
                                    eta_MT_4( i, j ) * eta_V1( m, n ) ) ...
                                    / sum_mu_aff_4_MT;

                        end;
                    end;
                end;
            end;
        end;

        clear i j num sum_mu_aff_ON_MT sum_mu_aff_OFF_MT startm endm ...
              startn endn old_mu_aff_ON_MT old_mu_aff_OFF_MT;

        % Update MT lateral weights.
        for ( i = 1:MT_d )
            for ( j = 1:MT_d )

                sum_E_MT = 0;
                sum_I_MT = 0;
                num = (i-1)*MT_d+j;
```

```
1546                startm = max( 1,   i - (r_E-0.5) );
1547                endm   = min( MT_d, i + (r_E-0.5) );
1548                startn = max( 1,   j - (r_E-0.5) );
1549                endn   = min( MT_d, j + (r_E-0.5) );
1550
1551            %            if( endm > N_d ) endm = N_d; end
1552            %            if( endn > N_d ) endn = N_d; end
1553
1554                for ( m=startm:endm )
1555                    for ( n = startn:endn )
1556                            if ( E_circle(m-startm+1,n-startn+1 ) )
1557                                sum_E_MT = sum_E_MT + old_E_MT( num, ...
1558                                m-startm+1, n-startn+1 ) + alpha_E * ...
1559                                eta_MT( i, j ) * eta_MT( m, n );
1560                            end;
1561                    end;
1562                end;
1563
1564               for ( m = startm:endm )
1565                    for ( n = startn:endn )
1566                        if ( E_circle(m-startm+1,n-startn+1 ) )
1567                            E_MT( num, m-startm+1, n-startn+1 ) = ...
1568                                    (old_E_MT( num, m-startm+1, ...
1569                                    n-startn+1 ) + alpha_E * ...
1570                                    eta_MT( i, j ) * eta_MT( m, n )) ...
1571                                    / sum_E_MT;
1572                        else
1573                            E_MT(num,m-startm+1,n-startn+1 ) = 0;
1574                        end;
1575                    end;
1576                end;
1577
1578                startm = max( 1, i - (r_I-0.5) );
1579                endm   = min( MT_d, i + (r_I-0.5) );
1580                startn = max( 1, j - (r_I-0.5) );
1581                endn   = min( MT_d, j + (r_I-0.5) );
1582
1583                for ( m=startm:endm )
1584                    for ( n = startn:endn )
1585                        if ( I_circle(m-startm+1,n-startn+1 ) )
1586                           sum_I_MT = sum_I_MT + old_I_MT( num, ...
1587                                       m-startm+1, n-startn+1 ) + ...
1588                                       alpha_I * eta_MT( i, j ) * ...
1589                                       eta_MT( m, n );
1590                       end;
1591                    end;
1592                end;
1593
```

```
1594
1595
1596                for ( m = startm:endm )
1597                    for ( n = startn:endn )
1598                        if ( I_circle(m-startm+1,n-startn+1 ) )
1599                           I_MT( num, m-startm+1, n-startn+1 ) = ...
1600                                    (old_I_MT( num, m-startm+1, ...
1601                                    n-startn+1 ) + alpha_I * ...
1602                                    eta_MT( i, j ) * eta_MT( m, n )) ...
1603                                    / sum_I_MT;
1604                           end;
1605                    end;
1606                end;
1607
1608
1609            end;
1610        end;
1611
1612        % Finish updating lateral weights (and all weights).
1613
1614        clear i j m n num old_E_MT startm endm startn endn ...
1615              sum_I_MT sum_E_MT;
1616        end; % End settling iterations.
1617
1618        clear blah;
1619        pack;
1620
1621        unknown_response_MT = eta_MT - old_eta_MT;
1622
1623
1624
1625        %
1626        % Display the results for this round to the user.
1627        %
1628        imagesc( 1-xi_REC , 'Tag','test','Parent', Input_h, ...
1629                    'CDataMapping', 'scaled', [0 100]  );
1630
1631        imagesc( 1-(eta_LGN_ON - eta_LGN_OFF) , 'Tag','test','Parent', ...
1632                    LGN_h, 'CDataMapping', 'scaled', [0 100]  );
1633
1634        imagesc( eta_V1 , 'Tag','test','Parent', V1_h, 'CDataMapping', ...
1635                    'scaled', [0 100]  );
1636
1637        imagesc( unknown_response, 'Tag', 'test', 'Parent', ...
1638                    difference_V1_h, 'CDataMapping', 'scaled', [0 100] );
1639
1640        imagesc( eta_MT, 'Tag', 'test', 'Parent', MT_h, 'CDataMapping', ...
1641                    'scaled', [0 100] );
```

```
1642
1643            imagesc( unknown_response_MT, 'Tag', 'test', 'Parent', ...
1644                        difference_MT_h, 'CDataMapping', 'scaled', [0 100] );
1645
1646
1647
1648
1649
1650
1651            %FOR STATISTICAL ANALYSIS
1652                        for ( i = 1:N_d )
1653                                for ( j = 1:N_d )
1654                                        num = (i-1)*N_d+j;
1655                    testerV1(i,j,1) = sum(sum(abs(old_I_V1( num, :, : )   ...
1656                                                        I_V1( num, :, : ))));
1657                                end;
1658                        end;
1659            for ( i = 1:MT_d )
1660                                for ( j = 1:MT_d )
1661                                        num = (i-1)*MT_d+j;
1662                        testerMT(i,j,1) = sum(sum(abs(old_I_MT( num, :, : ) ...
1663                                                    - I_MT( num, :, : ))));
1664                                end;
1665                        end;
1666            clear i j num old_I_V1;
1667            %END STATISTICAL ANALYSIS
1668
1669            sum_V1 = [sum_V1;sum( sum( testerV1(:,:,1) ) )];
1670            sum_MT = [sum_MT;sum( sum( testerMT(:,:,1) ) )];
1671
1672            if( t > 300 )
1673                show_sum_V1 = sum_V1(t-299:t);
1674                show_sum_MT = sum_MT(t-299:t);
1675            else
1676                show_sum_V1 = sum_V1;
1677                show_sum_MT = sum_MT;
1678            end;
1679
1680            plot( show_sum_V1, 'Parent', V1Graph_h );
1681            plot( show_sum_MT, 'Parent', MTGraph_h );
1682
1683            pause( 1 );
1684
1685            t = t + 1;
1686
1687            %
1688            % Save the file every 100 iterations, on the first iteration
1689            % and if the save button is pressed
```

```
            %
            if( mod( t, 100  ) == 0 )
                test_image = xi_REC;
                name = strcat( filename, ' ', num2str( t ), '.mat' );
                save( [pathname, name] );
            end;

            if( training == 0 )
                disp( 'Saving data...' )
                test_image = xi_REC;
                name = strcat( filename, '_', num2str( t ), '.mat' );
                save( [pathname, name] );
                disp( 'Data Saved' )
            end;

            if( saveFile == 1 )
                disp( 'Saving data...' )
                test_image = xi_REC;
                name = strcat( filename, '_', num2str( t ), '.mat' );
                save( [pathname, name] );
                disp( 'Data Saved' )
            end;

            pack;

                end;

        if( training == 0 )
            pause( 2 );
        end;

    end;

end;
```

A.4 Helper Functions

```
function [pathname,filename] = LoadData( )
%
% Holds the different method calls depending on the processing
% requested.  X is the image chosen and Y is the method.
%

[filename,pathname] = uigetfile('*.mat','Select the MAT Data Set');
if isequal(filename,0)|isequal(pathname,0)
```

```
    disp('File not found')
else
    disp(['File ', pathname, filename, ' found...'])
    disp('Loading Data...')
end
```

```
function [pathname,filename] = SaveData( )
%
% Holds the different method calls depending on the processing
% requested.  X is the image chosen and Y is the method.
%

[filename,pathname] = uiputfile('*.mat','Select the VIS Data Set');
if isequal(filename,0) | isequal(pathname,0)
    disp('No file specified')
else
    disp(['File ', pathname, filename, ' created...'])
    disp('Saving Data...')
end
```

```
% Calculates the obvious repeating portion of
% the formula 4.1 on page 37 of the Bednar thesis.

function ret=normgaus( X, X0, Y, Y0, SIGMA )


ret = ( X   X0 )  * ( X - X0 );
ret = ret + ( ( Y - Y0 ) .* ( Y - Y0 ) );
ret = ret / ( SIGMA * SIGMA );
ret = ret .* -1;
ret = exp( ret );
```

```
%
% Will simply return a circle mask given a radius
%

function ret=draw_circle( radius )


x = ones(1, (radius*2) )' * [(radius):-1:0.5,1.5:1:radius];
y = x';
ret = (((x.^2+y.^2).^0.5)<=(radius+0.5));
```

A.5 Rainbow Colormaps

```
function [cmap] = rainbow(maplength);
% ColEdit function [cmap] = rainbow(maplength);
%
% colormap m-file written by ColEdit
% version 1.0 on 25-Apr-2003
%
% input  :        [maplength]        [64]        - colormap length
%
% output :        cmap                           colormap RGB-value array

% set red points
r = [ [];...
    [0 0];...
    [0.11282 1];...
    [0.24872 0.98936];...
    [0.36923 0.98936];..
    [0.5 0.12766];...
    [0.62564 0.28723];...
    [0.75385 0.021277];...
    [0.88462 0.35106];...
    [1 0.5];...
    [] ];

% set green points
g = [ [];...
    [0 0];..
    [0.11795 0.010638];...
    [0.24872 0.59574];..
    [0.3641 0.97872];...
    [0.5 0.60638];...
    [0.61795 0.55319];...
    [0.75128 0.32979];...
    [0.87692 0.12766];...
    [1 0.042553];...
    [] ];

% set blue points
b = [ [];...
    [0 0.010638];...
    [0.11795 0.010638];...
    [0.24872 0.010638];...
    [0.37949 0.010638];..
    [0.5 0.15957];...
    [0.63333 0.48936];...
    [0.74872 0.52128];...
```

```
46      [0.88205 0.57447];...
47      [1 0.48936];...
48      [] ];
49  % ColEditInfoEnd
50
51  % get colormap length
52  if nargin==1
53    if length(maplength)==1
54      if maplength<1
55        maplength = 64;
56      elseif maplength>256
57        maplength = 256;
58      elseif isinf(maplength)
59        maplength = 64;
60      elseif isnan(maplength)
61        maplength = 64;
62      end
63    end
64  else
65    maplength = 64;
66  end
67
68  % interpolate colormap
69  np = linspace(0,1,maplength);
70  rr = interp1(r(:,1),r(:,2),np,'linear');
71  gg = interp1(g(:,1),g(:,2),np,'linear');
72  bb = interp1(b(:,1),b(:,2),np,'linear');
73
74  % compose colormap
75  cmap = [rr(:),gg(:),bb(:)];
```

```
1   function [cmap] = rainbow2(maplength);
2   % ColEdit function [cmap] = rainbow2(maplength);
3   %
4   % colormap m-file written by ColEdit
5   % version 1.0 on 12-May-2003
6   %
7   % input  :        [maplength]        [64]          colormap length
8   %
9   % output :        cmap                          - colormap RGB-value array
10
11  % set red points
12  r = [ [];...
13      [0 0];...
14      [0.012821 0.97872];...
15      [0.079487 0.97872];...
16      [0.13846 0.97872];...
17      [0.20513 0.97872];...
```

```
18      [0.25897 0.96809];...
19      [0.32051 0.95745];...
20      [0.4 0.97872];...
21      [0.48462 0.68085];...
22      [0.5359 0.5];...
23      [0.59744 0.12766];...
24      [0.66154 0.17021];...
25      [0.72308 0.18085];...
26      [0.79231 0.085106];...
27      [0.84615 0.26596];...
28      [0.92308 0.34043];...
29      [0.97436 0.51064];...
30      [1 0.97872];...
31      [] ];
32
33  % set green points
34  g = [ [];...
35      [0 0.074468];...
36      [0.012821 0.010638];...
37      [0.082051 0.24468];...
38      [0.14103 0.42553];...
39      [0.19487 0.48936];...
40      [0.25641 0.69149];...
41      [0.31795 0.85106];. .
42      [0.41538 0.97872];...
43      [0.59231 0.98936];...
44      [0.65897 0.61702];...
45      [0.69744 0.38298];...
46      [0.79487 0.11702];...
47      [0.85385 0.12766];...
48      [0.90513 0.1383];...
49      [0.98205 0.074468];...
50      [1 0.96809];...
51      [] ];
52
53  % set blue points
54  b = [ [];...
55      [0 0.021277];. .
56      [0.020513 0.021277];...
57      [0.076923 0.010638];...
58      [0.12564 0.021277];...
59      [0.18462 0.021277];...
60      [0.5 0.010638];...
61      [0.61538 0.042553];..
62      [0.67949 0.28723];...
63      [0.74359 0.54255];...
64      [0.79744 0.98936];...
65      [0.85385 0.7766];...
```

```
66      [0.90256 0.61702];...
67      [0.97692 0.5];...
68      [1 0.98936];...
69      [] ];
70  % ColEditInfoEnd
71
72  % get colormap length
73  if nargin==1
74    if length(maplength)==1
75      if maplength<1
76        maplength = 64;
77      elseif maplength>256
78        maplength = 256;
79      elseif isinf(maplength)
80        maplength = 64;
81      elseif isnan(maplength)
82        maplength = 64;
83      end
84    end
85  else
86    maplength = 64;
87  end
88
89  % interpolate colormap
90  np = linspace(0,1,maplength);
91  rr = interp1(r(:,1),r(:,2),np,'linear');
92  gg = interp1(g(:,1),g(:,2),np,'linear');
93  bb = interp1(b(:,1),b(:,2),np,'linear');
94
95  % compose colormap
96  cmap = [rr(:),gg(:),bb(:)];
```

List of Abbreviations

Abbreviation	Details
HLLISOM	Hierarchical LISSOM
LISSOM	Laterally Interconnected Synergetically Self-Organizing Map
PGO	ponto-geniculo-occipital
REM	rapid eye movement
RF-LISOM	the Receptive-Field LISSOM
SOM	Self Organizing Map
V1	Primary Visual Cortex

Bibliography

Beardsley, S.A. & Vaina, L.M. (1998). Computational modelling of optic flow selectivity in mstd neurons. *Network: Computation in Neural Systems*, **9**, 467–493.

Bednar, J. (2002). *Learning to See: Genetic and Environmental Influences on Visual Development*. Ph.D. thesis, University of Texas at Austin.

Bednar, J.A. & Miikkulainen, R. (2003). Self-organization of spatiotemporal receptive fields and laterally connected direction and orientation maps. *Neurocomputing*, **52-54**, 473–480.

Blake, R., Sekuler, R. & Grossman, E. (2003). *Motion Processing in Human Visual Cortex*. CRC Press, Boca Raton, Florida.

Blakemore, C. & Cooper, G. (1970). Development of the brain depends on the visual environment. *Nature*, **228**, 477–478.

Brazeau, J.D., Humphrey, J. & Gaborski, R. (2003). A biologically-based model of a visual learning system. In *2003 Western New York Image Processing Workshop*, 55–58.

Chalupa, L.M. & Rhoades, R.W. (1988). Environmental influences upon the

development of the golden hamster's visual system. *Neurophysiology and psychophysiology: Experimental and clinical applications*, 889–935.

Chapman, B. (2000). Necessity for afferent activity to maintain eye-specific segregation if ferret lateral geniculate nucleus. *Science*, **287**, 2479–2483.

Chapman, B. & Stryker, M.P. (1993). Development of orientation selectivity in ferret primary visual cortex and effects of deprivation. *Journal of Neuroscience*, **13**, 5251–5262.

Coon, D. (2000). *Essentials of Psychology: Exploration and Application*. Wadsworth, Belmont, Clifornia, eighth edn.

Crair, D.C., M. C.and Gillespie & Stryker, M.P. (1998). The role of visual experience in the development of columns in cat visual cortex. *Science*, **279**, 566–570.

Gilbert, C.D., Hirsch, J. & Wiesel, T. (1990). Lateral interaction in visual cortex. *Cold Spring Harbor Symposia on Quatitative Biology*, **LV**, 663–677.

Grossberg, S., Mingolla, E. & Viswanathan, L. (2001). Neural dynamics of motion integration and segmentation within and across apertures. *Vision Research*, **41**, 2521–2553.

Grzywacz, N.M. & Merwine, D. (2002). *Motion Perception, Neural Basis of*. Macmillan Press, Cambridge.

Hebb, D.O. (1949). *The Organization of Behavior*. John Wiley and Sons, New York.

Kandel, E.R., Schwartz, J. & Jessell, T. (1991). *Principles of Neural Science*. McGraw-Hill, New York, 3rd edn.

Kasamatsu, T., Kitano, M., Sutter, E. & Norcia, A. (1998). Lack of lateral inhibitory interaction in visual cortex of monocularly deprived cats. *Vision Research*, **38**, 1–12.

Katz, L.C. & Shatz, C.J. (1996). Synaptic activity and the construction of cortical circuits. *Science*, **274**, 1133–1138.

Kohonen, T. (1997). *Self-Organizing Maps*. Springer, 2nd edn.

Linsker, R. (1986). From basic network principles to neural architecture: Emergence of orientation columns. *Proceedings of the National Academy of Sciences*, **83**, 8779–8783.

Lowel, S. & Singer, W. (1992). Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science*, **255**, 209–212.

Marks, G.A., Shaffery, J.P., Oksenberg, A., Speciale, S.G. & Roffwarg, H.P. (1995). A functional role for rem sleep in brain maturation. *Behavioural Brain Research*, **69**, 1–11.

Medler, D. (1998). A brief history of connectionism. Tech. rep., University of Alberta.

Newsome, W.T. & Pare, E.B. (1998). A selective impairment of motion perception following lesions of the middle temporal visual area (mt). *Journal of Neuroscience*, **8**, 2201–2211.

Piepenbrock, C., Ritter, H. & Obermayer, K. (1996). Cortical map development driven by spontaneous retinal activity waves. In *ICANN*, 427–432.

Pomplun, M., Liu, Y., Martinez-Trujillo, J., Simine, E. & Tsotsos, J. (2002). A neurally-inspired model for detecting and localizing simple motion patterns in image sequences. In *4th Workshop Dynamic Perception*, 47–52.

Rochester, N., Holland, J.H., Haibt, L.H. & Duda, W.L. (1956). Tests on a cell assembly theory of the action of the brain, using a large digital computer. *IRE Transactions on Information Theory*, **2**, 80–93.

Rolls, E.T. & Deco, G. (2002). *Computational Neuroscience of Vision*. Oxford University Press, Oxford.

Rosenzweig, Leiman & Breedlove (1999). *Bioloical Psychology: An Introduction to Behavioral, Cognitive, and Clinical Neuroscience*. Sinauer Associates, Inc, Sunderland, Massachusetts, 2nd edn.

Simoncelli, E.P. & Heeger, D.J. (1998). A model of neuronal responses in visual area mt. *Vision Research*, **38**, 743–761.

Sirosh, J. (1995). *A Self-Organizing Neural Network Model of the Primary Visual Cortex*. Ph.D. thesis, University of Texas at Austin.

Stratton, G. (1897). Vision without inversion of the retinal image. *Psychological Review*, **4**, 341–360.

Thiele, A., Dobkins, K.R. & Albright, T.D. (1999). The contribution of color to motion processing in macaque middle temporal area. *The Journal of Neuroscience*, **19**, 6571–6587.

von der Malsburg, C. (1973). Self-organization of orientation-sensitive cells in the straiate cortex. *Kybernetik*, **15**, 85–100.

Wiesel, T.N. (1982). Postnatal development of the visual cortex and the influence of environment. *Nature*, **299**, 583–591.