LIDS-P-2186

Likelihood Calculation for a Class of Multiscale Stochastic Models, with Application to Texture Discrimination[1]

Mark R. Luettgen[2]

Alan S. Willsky[3]

Abstract

A class of multiscale stochastic models based on scale-recursive dynamics on trees has recently been introduced. Theoretical and experimental results have shown that these models provide an extremely rich framework for representing both processes which are intrinsically multiscale, e.g., $1/f$ processes, as well as 1-D Markov processes and 2-D Markov random fields. Moreover, efficient optimal estimation algorithms have been developed for these models by exploiting their scale-recursive structure. In this paper, we exploit this structure in order to develop a computationally efficient and parallelizable algorithm for likelihood calculation. We illustrate one possible application to texture discrimination and demonstrate that likelihood-based methods using our algorithm have substantially better probability of error characteristics than well-known least-squares methods, and achieve performance comparable to that of Gaussian Markov random field based techniques, which in general are prohibitively complex computationally.

Edics classification: IP 1.6

[1]This work was supported by the Air Force Office of Scientific Research under grant AFOSR-92-J-0002, by the Office of Naval Research under Grant N00014-91-J-1004, and by the Advanced Research Projects Agency under contract DAAH01-93-C-R021 and through Air Force Grant F49620-93-1-0604.

[2]Alphatech, Inc., 50 Mall Road, Burlington, MA 01803. Electronic mail address: luettgen@alphatech.com.

[3]Room 35-433, MIT Laboratory for Information and Decision Systems, Cambridge, MA 02139. Electronic mail address: willsky@mit.edu.

1 Introduction

A class of multiscale models describing stochastic processes indexed by the nodes of a tree has recently been introduced in [4, 5]. This class of processes is quite rich. In particular, experimental results in [4] illustrate that these models are able to capture the statistical self-similarity exhibited by stochastic processes with generalized power spectra of the form $1/f^{\beta}$. Moreover, in [12] we have described how they can be used to represent *any* 1-D Markov process or 2-D Markov random field.

The basic concept underlying this modeling framework is the exploitation of the time-like nature of *scale*. In particular, these models provide a scale-recursive description for random processes and fields and, as a result, lead to extremely efficient scale-recursive algorithms for optimal estimation [4, 5]. In particular, while standard 2-D optimal estimation formulations — e.g., those based on MRF's — have per-pixel computational complexities that typically grow with image size, our scale-recursive algorithms have a per-pixel complexity independent of image size and thus can lead to substantial computational savings for standard image processing problems [13].

The conclusion that we draw from this is that the multiscale framework can in many cases provide a very useful basis for signal and image processing problems, both because of the rich class of phenomena that it can be used to describe and because of the efficient algorithms to which it leads. This motivates further algorithmic development and, in particular, we discuss in this paper a likelihood calculation algorithm for this class of processes. That is, we consider the problem of computing the log of the probability density of a set of noisy observations assuming that the data corresponds to a particular multiscale model. We exploit the structure of the multiscale models to develop an efficient and parallelizable algorithm that allows for multiresolution data and parameters which vary in both space and scale. The algorithm is non-iterative and again has a constant per-pixel computational complexity.

We illustrate one possible application of the algorithm to a texture classification problem in

which one must choose from a given set of models that model which best represents or most likely corresponds to a given set of random field measurements [9]. Texture modeling with Gaussian Markov random field (GMRF) models is well documented in the literature [1, 6, 16]. One difficulty in using GMRF models, however, is that the calculation of likelihoods can be prohibitively complex computationally. If data are available on a regular rectangular grid, likelihoods for stationary GMRF's can be computed efficiently using 2-D FFT's. However, if there is an irregular sampling pattern or if there are regions without data (due to camera blockage, for instance) then the 2-D FFT approaches break down for GMRF models and exact likelihood calculation becomes computationally infeasible for even moderately sized domains.

As developed in [12], multiscale models representing GMRF's to any desired level of fidelity can be readily constructed and this immediately suggests the idea of developing texture models and discrimination algorithms based on the multiscale modeling framework and the associated likelihood calculation algorithm that we develop in this paper. However, it is not immediately obvious that such a framework will provide significant advantages over classical GMRF-based approaches. Specifically, the approach developed in [12] yields a family of multiscale models representing approximations of a GMRF of increasing fidelity *and* complexity. Thus, if we require exact modeling of the GMRF, the apparent computational gain in using the multiscale framework may have diminished to the point that the benefit of our formalism is not particularly significant. However, as the results in [12] illustrate, there is strong evidence that relatively low-order models yield processes which are visually indistinguishable from realizations of the GMRF's they approximate. In this paper we show that a corresponding statement is true when low-order multiscale models are used in place of GMRF priors as the basis for algorithm design, in this case for texture discrimination. Indeed, as we will see, we can achieve essentially the same performance in discriminating between two GMRF textures using likelihood calculations based on low-order multiscale models as can be achieved using

exact likelihoods for the GMRF models. Since for these low-order models the likelihood calculation algorithm is extremely efficient, and since the algorithm allows for arbitrarily irregular sampling patterns (i.e. it applies in many practical situations in which GMRF-based approaches relying on 2-D FFT computations do not), what this shows is that the multiscale framework *does* in fact offer substantial advantages over the GMRF-based framework. Indeed, given the potentially substantial computational savings in using the multiscale approach, and the fact that any model for a real texture is an idealization, these results demonstrate a potential advantage in using multiscale models, rather than GMRF's, as a valid starting point for the modeling of textures.

This paper is organized as follows. In Section 2 we discuss the class of multiscale stochastic models and the scale-recursive estimation algorithm associated with them. In Section 3 we present the algorithm for performing likelihood calculations. In Section 4 we present results of experiments that demonstrate the relative performance of our multiscale approach and GMRF-based approaches to texture discrimination. Finally, our conclusions are summarized in Section 5.

2 Multiscale Stochastic Modeling and Optimal Estimation

2.1 Multiscale Stochastic Models

The models presented in this section describe multiscale Gaussian stochastic processes indexed by nodes on a *tree*. A q^{th} order tree is a pyramidal structure of nodes connected such that each node of the tree has q offspring. Different levels of the tree correspond to different scales of the process. In particular, the q^m values at the m^{th} level of the tree are interpreted as "information" about the m^{th} scale of the process. For instance, quadtree models naturally arise in 2-D applications, and the simplest example of a quadtree multiscale representation is that in which the values of the spatial process at the m^{th} scale correspond to averages of the process values at scale $m+1$ [15, 13].

However, the state variables can also be used to represent many other properties of the process of interest. For example, in [12], in which we demonstrate that these multiscale models can be used to represent any 1-D Markov process or 2-D MRF, the state variables at coarser scales are interpreted as decimated, rather than averaged, versions of the process at the finest scale. On the other hand, the *approximate* representations of GMRF's developed in [12] and that we use in Section 4 are based on yet another interpretation which is associated with both averaging *and* decimation.

An example of a q^{th}-order tree (for $q = 3$) is depicted in Figure 1. Here, each horizontal level corresponds to a particular scale, with coarser scales toward the top of the tree and finer scales toward the bottom. We denote nodes on the tree with an abstract index s, and define an upward (fine-to-coarse) shift operator $\bar{\gamma}$ such that $s\bar{\gamma}$ is the parent of node s. We also define a corresponding set of downward (coarse-to-fine) shift operators $\alpha_i, i = 1, 2, \cdots, q$, such that the q offspring of node s are given by $s\alpha_1, s\alpha_2, \cdots, s\alpha_q$, and we let $m(s)$ denote the level or scale of the node s (so that $m(s\bar{\gamma}) = m(s) - 1$ and $m(s\alpha_i) = m(s) + 1$). Finally, we define the operator $\bar{\delta}$ such that if $s = s\bar{\gamma}\alpha_k$, then $s\bar{\delta} = s\bar{\gamma}\alpha_{k+1}$, with the convention that $\alpha_{q+1} \equiv \alpha_1$. In words, $\bar{\delta}$ is a *horizontal* shift operator, defined cyclically, and such that if s is the k^{th} offspring of its parent, then $s\bar{\delta}$ corresponds to the $(k+1)(\text{mod } q)^{th}$ offspring of the same parent node.

The multiscale stochastic models of interest here are specified in terms of scale-recursive dynamic equations defined on the tree. Specifically, let $x(s) \in \mathcal{R}^n$ denote the value of the "state" of the process at node s. The statistical characterization of $x(s)$ is then given by:

$$x(s) = A(s)x(s\bar{\gamma}) + B(s)w(s) \tag{1}$$

under the assumptions that $x(0) \sim \mathcal{N}(0, P(0))$, $w(s) \sim \mathcal{N}(0, I)$, $A(s)$ and $B(s)$ are matrices of appropriate size, and $s = 0$ corresponds to the root node at the top of the tree[4]. The state variable $x(0)$ provides an initial condition for the recursion. The driving noise $w(s) \in \mathcal{R}^m$ is white, i.e. $w(s)$

[4]The notation $x \sim \mathcal{N}(m, P)$ means that x is normally distributed with mean vector m and covariance P.

and $w(\sigma)$ are independent if $s \neq \sigma$, and independent of the initial condition. Interpreting each level as a representation of one scale of the random process or field of interest, we see that (1) describes its evolution from coarse to fine scales. The term $A(s)x(s\bar{\gamma})$ represents interpolation or prediction down to the next level, and $B(s)w(s)$ represents new information or detail added as the process evolves from one scale to the next.

The class of models (1) has a statistical structure that we can exploit to develop extremely efficient algorithms. In particular, note that any given node on the q^{th}-order tree can be viewed as a boundary between $q+1$ subsets of nodes (q corresponding to paths leading towards offspring and one corresponding to a path leading towards a parent). An important property of the scale-recursive model (1) is that not only is it Markov from scale-to-scale but also, conditioned on the value of the state at any node, the values of the state corresponding to the $q+1$ corresponding subsets of nodes are independent. This fact is the basis for the development in [4, 5] of an algorithm for computing smoothed estimates of $x(s)$ based on noisy measurements $y(s) \in \mathcal{R}^p$ of the form:

$$y(s) = C(s)x(s) + v(s) \tag{2}$$

where $v(s) \sim \mathcal{N}(0, R(s))$, is independent of both the driving noise $w(s)$ and the initial condition $x(0)$, and the matrix $C(s)$ specifies measurements taken at different spatial locations and perhaps at different scales. This algorithm provides the starting point for our likelihood calculation algorithm, and hence we briefly review it in the next section.

2.2 Multiscale Optimal Estimation

The algorithm for computing the smoothed estimates of $x(s)$ consists of an upward sweep in which the available measurement information in a subtree is successively fused in a fine-to-coarse recursion in scale, followed by a downward sweep in which the information is spread back throughout the tree. We denote the set of measurements in the subtree which has s as its root as Y_s, i.e. $Y_s =$

$\{y(\sigma)|\sigma = s$ or σ is a descendant of $s\}$. We also define $\hat{x}(s|Y)$ as the expected value of the state $x(s)$ given measurements in the set Y, i.e. $\hat{x}(s|Y) = \mathbf{E}[x(s)|Y]$. The set Y can be any subset of the measurements on the tree. In particular, the smoothed estimate of $x(s)$, the estimate based on all of the data, is denoted $\hat{x}(s|Y_0)$. Finally, we define the error covariance corresponding to $\hat{x}(s|Y)$ as $P(s|Y) = \mathbf{E}[(x(s) - \hat{x}(s|Y))(x(s) - \hat{x}(s|Y))^T]$, and the set $Y_s^{\alpha_q} = Y_s \setminus \{y(s)\}$, where the notation $Y_s \setminus \{y(s)\}$ means that the measurement $y(s)$ is not included in the set $Y_s^{\alpha_q}$. Note that $\hat{x}(s|Y_s^{\alpha_q})$ is the best estimate at node s given all of the data in the subtree strictly below node s, whereas $\hat{x}(s|Y_s)$ is the best estimate including $y(s)$ as well. The upward sweep of the smoothing algorithm computes these quantities recursively from fine-to-coarse scales. The initialization of $\hat{x}(s|Y_s^{\alpha_q})$ and the corresponding error covariance $P(s|Y_s^{\alpha_q})$ at the finest level reflect the prior statistics of $x(s)$ at the finest scale, as we have not yet incorporated data. In particular, for every s at this finest scale we set $\hat{x}(s|Y_s^{\alpha_q})$ to zero (which is the prior mean of $x(s)$) and similarly set $P(s|Y_s^{\alpha_q})$ to the corresponding covariance, namely the solution at the finest level of the Lyapunov equation:

$$P(s) = A(s)P(s\bar{\gamma})A^T(s) + B(s)B^T(s) \tag{3}$$

where $P(s)$ denotes the variance of the process $x(s)$ at node s. The upward sweep of the smoothing algorithm then proceeds recursively. Specifically, suppose that we have $\hat{x}(s|Y_s^{\alpha_q})$ and $P(s|Y_s^{\alpha_q})$ at a given node s. Then this estimate is *updated* to incorporate the measurement $y(s)$ (if there is a measurement at node s) according to the following:

$$\hat{x}(s|Y_s) = \hat{x}(s|Y_s^{\alpha_q}) + K(s)[y(s) - C(s)\hat{x}(s|Y_s^{\alpha_q})] \tag{4}$$

$$P(s|Y_s) = [I - K(s)C(s)]P(s|Y_s^{\alpha_q}) \tag{5}$$

where $K(s) = P(s|Y_s^{\alpha_q})C^T(s)[C(s)P(s|Y_s^{\alpha_q})C^T(s) + R(s)]^{-1}$.

Suppose then that we have the updated estimates $\hat{x}(s\alpha_i|Y_{s\alpha_i})$ at all of the immediate descendants of node s. The next step involves the use of these estimates to predict $x(s)$ at the next coarser

scale, i.e. to compute $\hat{x}(s|Y_{s\alpha_i})$. Specifically, we can define an upward model for the tree process which describes its evolution in terms of fine-to-coarse dynamics [5, 4]:

$$x(s\bar{\gamma}) = F(s)x(s) + \bar{w}(s) \tag{6}$$

with the measurement equation again given by (2), and where $F(s) = P(s\bar{\gamma})A^T(s)P(s)^{-1}$ and $\mathbf{E}[\bar{w}(s)\bar{w}^T(s)] = P(s\bar{\gamma}) - P(s\bar{\gamma})A^T(s)P(s)^{-1}A(s)P(s\bar{\gamma}) \equiv Q(s)$. This upward model is equivalent to the downward model in the sense that the joint second order statistics of the states $x(s)$ and measurements $y(s)$ are the same. The driving noise term $\bar{w}(s)$ is white along any path from the finest to coarsest scales and, as a result, (6) can be used to obtain the fine-to-coarse predicted estimates:

$$\hat{x}(s|Y_{s\alpha_i}) = F(s\alpha_i)\hat{x}(s\alpha_i|Y_{s\alpha_i}) \tag{7}$$

$$P(s|Y_{s\alpha_i}) = F(s\alpha_i)P(s\alpha_i|Y_{s\alpha_i})F^T(s\alpha_i) + Q(s\alpha_i) \tag{8}$$

Next, note that $Y_s^{\alpha_q} = Y_{s\alpha_1} \cup Y_{s\alpha_2} \cup \cdots \cup Y_{s\alpha_q}$. This implies that $\hat{x}(s|Y_s^{\alpha_q})$ can be obtained by using standard formulas for combining linear least squares estimates based on independent sets of measurements in the following *merge* step:

$$\hat{x}(s|Y_s^{\alpha_q}) = P(s|Y_s^{\alpha_q})\sum_{i=1}^{q} P^{-1}(s|Y_{s\alpha_i})\hat{x}(s|Y_{s\alpha_i}) \tag{9}$$

$$P(s|Y_s^{\alpha_q}) = [(1-q)P(s)^{-1} + \sum_{i=1}^{q} P^{-1}(s|Y_{s\alpha_i})]^{-1} \tag{10}$$

The upward sweep given by the update, predict and merge equations proceeds recursively up the tree. At the top of the tree (corresponding to the root node $s = 0$), one obtains the *smoothed* estimate of the root node, $\hat{x}(0|Y_0)$. The estimate $\hat{x}(0|Y_0)$ provides initialization for a *downward sweep* in which $\hat{x}(s|Y_0)$ is computed recursively from coarse-to-fine. It is also possible to derive a recursion for the smoothing error covariance $P(s|Y_0)$, and in fact a multiscale model for the smoothing error process which allows one to calculate the full correlation structure of the error statistics. The reader is referred to [4, 11, 14] for further details. Our primary focus here will be on

the relationship of the upward sweep of the algorithm, which we refer to as the multiscale Kalman filtering algorithm, to the likelihood calculations discussed in the next section.

3 Likelihood Function Calculation

In this section, we provide an algorithm for computing the likelihood function for the multiscale model, i.e. the log of the probability density of the data based on this model. We denote the set of nodes on the tree at which we have measurements as $\mathcal{T}$ and stack the measurements $\{y(s)\}_{s\in\mathcal{T}}$ into a vector y. Then, $y \sim \mathcal{N}(0, \Lambda_y)$, where Λ_y is implicitly given by the model parameters.

The main problem in evaluating the likelihood is that the data covariance matrix Λ_y is generally full, and thus inverting it directly is difficult if the number of data points is large. The algorithm below *whitens* the data, which allows the likelihood to be evaluated easily. In particular, the data is invertibly transformed to a new set of data $\{\nu(s)\}_{s\in\mathcal{T}}$, such that $\nu(s)$ and $\nu(\sigma)$ are uncorrelated if $s \neq \sigma$. In particular, if we construct a vector ν by stacking up the residuals $\{\nu(s)\}$, then $\nu = Ty$ for some invertible matrix T and the resulting covariance matrix, $\Lambda_\nu = T\Lambda_y T^T$, is diagonal (or block diagonal)[5]. The likelihood function expressed in terms of $\{\nu(s)\}$ and its statistics is then given by:

$$\mathcal{L} = -\log|T| - \frac{m}{2}\log 2\pi - \frac{1}{2}\sum_{s\in\mathcal{T}}[\log|\Lambda_{\nu(s)}| + \nu^T(s)\Lambda^{-1}_{\nu(s)}\nu(s)] \tag{11}$$

where m is the dimension of y[6].

Achieving a computational gain via whitening the data depends upon finding a transformation T for which (a) the specification of T and the calculation $\nu = Ty$ can be performed efficiently and (b) $|T|$ = constant (usually equal to 1) independent of the parameters of the model[7]. One obvious choice for the transformation T is that based on an eigendecomposition of Λ_y. In this case (b)

[5]For simplicity in notation, we will use $\{\nu(s)\}$ in place of $\{\nu(s)\}_{s\in\mathcal{T}}$, and similarly $\{y(s)\}$ in place of $\{y(s)\}_{s\in\mathcal{T}}$.

[6]For example, if each measurement $y(s)$ is of dimension p, then $m = pS$, where S is the cardinality of $\mathcal{T}$.

[7]The latter can be of critical importance in parameter estimation since the dependence of $|T|$ on the parameters can greatly complicate maximization of $\mathcal{L}$ as a function of the parameters.

is trivially satisfied, but only in certain situations will the same be true for (a). For example, if the parameters of the model (1) vary only as a function of scale, then the Haar transform (and appropriate generalizations for trees of order $q > 2$) can be used to whiten the data [5]. On the other hand, if either the model or process is non-stationary (e.g., if the model parameters in (1) vary in both space and scale) or if the *data* collection is non-stationary (e.g., if the data for a 1-D or 2-D process has data dropouts, if 2-D data are available at a non-rectangular or irregular set of points, or if $C(s)$ in (2) depends fully on s and *not* just on $m(s)$) then not only is the determination of the eigenstructure of Λ_y extremely complex but also, even if we can compute an eigendecomposition, the calculation $\nu = Ty$ will itself be complex (in general, $\mathcal{O}(m^2)$ operations).

There are, of course, alternatives to the full eigendecomposition method of whitening. In particular, Gram-Schmidt orthogonalization, in which the data are ordered and sequentially whitened, provides another approach which also automatically satisfies (b)[8]. Satisfying (a), on the other hand, requires the availability of substantial structure. In particular, in general the whitening of each successive data point requires subtracting from it an estimate of it based on all data preceding it in the chosen ordering and without additional structure the computation of these estimates requires growing memory as the orthogonalization procedure proceeds. For 1-D time series, in which there is an obvious, natural ordering of the data points, the class of causal Gauss-Markov models has such structure and, in particular, the Kalman filter performs the whitening itself by generating the filter innovations, each sample of which is precisely the required difference between a measurement and its estimate based on previous data. The algorithm we present here can be viewed as a natural generalization to (1), (2) of the Kalman filter-based algorithm for 1-D Gauss-Markov models. In particular, note that for $q = 1$, the "tree" corresponds simply to a completely ordered sequence

[8]Assuming that y is constructed by stacking the measurements $\{y(s)\}$ in the desired order for whitening, we immediately see that Gram-Schmidt orthogonalization always yields a lower block triangular matrix T with identity blocks along the diagonal.

of points so that (1), (2) corresponds to the usual state space model for time series for which the Kalman filter performs the desired whitening. Our algorithm for general q^{th}-order trees is based directly on the multiscale Kalman filter described in Section 2, and hence reduces precisely to the standard algorithm in the case $q = 1$. For $q > 1$, the algorithm has an interesting new component not arising in the standard time series due to the fact that on higher order trees the multiscale Kalman filter provides only a *partial* whitening of the data.

In particular, define the residuals generated by the multiscale Kalman filter as $\nu_f(s) \equiv y(s) - C(s)\hat{x}(s|Y_s^{\alpha_q})$, where the subscript f is used to distinguished these filter residuals from the residuals $\nu(s)$ which will be the result of the likelihood calculation algorithm. The fact that the set $\{\nu_f(s)\}$ is not white for $q > 1$ is apparent from the update equation (4) in which the residual term $\nu_f(s)$ is used to obtain $\hat{x}(s|Y_s)$. Since the estimate $\hat{x}(s|Y_s^{\alpha_q})$ does not depend on nodes outside of the subtree below node s, there is no reason to expect that $\nu_f(s)$ is orthogonal to the corresponding residuals calculated at nodes at the same level as s. More generally, from the structure of the upward sweep we can immediately conclude that $\nu_f(s)$ and $\nu_f(\sigma)$ are necessarily uncorrelated if and only if one of these nodes is the ancestor of the other, i.e. if and only if for some $r > 0$, $s = \sigma\bar{\gamma}^r$ or $\sigma = s\bar{\gamma}^r$. Thus, along any single path from a fine scale node back toward the root node, the corresponding multiscale Kalman filter residuals are white. For usual time series corresponding to $q = 1$, there is only one such path. However, for $q > 1$, there are many paths and the Kalman filter, which operates in parallel on these, does not whiten the data across them. Nevertheless, the partial whitening that the Kalman filter performs can be taken advantage of and in the following subsections we describe an algorithm which does just that.

3.1 Ordering the Nodes

Any Gram-Schmidt procedure requires a total ordering of the data points. The essential attribute of the order we use is that offspring nodes appear earlier than their parents. This leads to a likelihood calculation algorithm which has a multiscale Kalman filter embedded in it, and which in essence provides the additional computations required to fully whiten the data.

To motivate the ordering scheme, recall that if we choose some path from finest to coarsest levels, the Kalman filter will produce a completely uncorrelated sequence along this path. Thus, if we take the residuals along one such path as elements of our final whitened process, we won't have any additional processing to do at these nodes. For example, consider the 3^{rd}-order tree in Figure 1 and suppose that we take the "left-most" path $s\alpha_1$–s–$s\bar{\gamma}$–0 as our chosen path along which we let the Kalman filter do all of the work — i.e. the corresponding Kalman filter residuals ν_f will also be the corresponding values of ν, our final, completely whitened process. Note first that the Kalman filter estimate is initialized with a value of zero at the finest level so that, referring to Figure 1, $\hat{x}(s\alpha_1|Y_{s\alpha_1}^{\alpha_q}) = 0$. Thus $\nu_f(s\alpha_1) = y(s\alpha_1)$, and it is natural then to think of the node $s\alpha_1$ in Figure 1 as the first point in our total ordering of the nodes. However, its parent s should *not* be thought of as the second point. In particular, while the Kalman filter residual $\nu_f(s)$ at this node has certainly been whitened with respect to $y(s\alpha_1)$, it has also been whitened with respect to $y(s\alpha_2)$ and $y(s\alpha_3)$, and thus, to take advantage of the work already performed by the Kalman filter, we should place $s\alpha_2$ and $s\alpha_3$ *before* s in our total order. Obviously, just as in our arbitrary choice of an initial path, we can arbitrarily choose to place one of these two points before the other, so we choose to order the first four points as $s\alpha_1, s\alpha_2, s\alpha_3, s$.

Continuing with this logic on Figure 1, since the Kalman filter will whiten $\nu_f(s\bar{\gamma})$ with respect to *all* of the data in the subtree beneath $s\bar{\gamma}$, all of these points should be placed before $s\bar{\gamma}$ in the order (so that $s\bar{\gamma}$ will be the 13^{th} point). Moreover, since the Kalman filter will also whiten $\nu(s\bar{\delta})$

with respect to its three descendants, those descendants should precede $s\bar{\delta}$ in the ordering. More generally, the ordering philosophy that this suggests is the following: Place each of the nodes as early in the order as possible, subject to the constraint that all descendants of any node precede that node in the order and to the constraint that each node *immediately* follows its descendent which is placed latest in the order (so that, referring to Figure 1 and the example above, node s follows node $s\alpha_3$ in the order). There is still some freedom in this ordering and we arbitrarily order the immediate offspring of any node s "left-to-right" as $s\alpha_1, s\alpha_2, \cdots, s\alpha_q$, as we have done in the example above. The resulting order for the tree in Figure 1 is given in Figure 2. For future reference, we now adopt the notation $s < \sigma$ if s appears before σ in this ordering.

With the ordering established in this way, we see that the Kalman filter does all of the work for some nodes but only part of it for others. For example, consider the tree of Figure 2. In this case, the Kalman filter will have done all of the desired whitening at nodes 1 (where no whitening is needed), 4 (whitened with respect to nodes 1 – 3), and 13 (whitened with respect to nodes 1 – 12). Thus at these three nodes we can take $\nu(s) = \nu_f(s)$. On the other hand, the Kalman filter does only part of the work for nodes 2 – 3, 5 – 8 and 9 – 12. For instance, node 8 is whitened relative to data at nodes 5 – 7 but *not* with respect to nodes 1 – 4. The key to performing the remaining whitening is to propagate information around the tree structure in an efficient way. In particular, what we wish to compute at any node s is $\mathbf{E}\{x(s)|y(\sigma), \sigma < s\}$, the estimate of the state $x(s)$ at that node based on measurements at all of the nodes preceding s in the ordering. Having this, the residual calculation is simply $\nu(s) = y(s) - C(s)\mathbf{E}\{x(s)|y(\sigma), \sigma < s\}$. The key then is to look carefully at the structure of the set of nodes $\{\sigma|\sigma < s\}$, and to perform the calculation using prediction, merge, and update steps analogous to those used in the Kalman filter.

3.2 Algorithm Description

Before describing the calculations required to obtain $\mathbf{E}\{x(s)|y(\sigma), \sigma < s\}$ in general, we first consider those corresponding to nodes 1 – 13 in Figure 2 in order to convey the essence of the algorithm. Consider first the whitening of the measurement at node 2 with respect to node 1 in Figure 2. As illustrated in Figure 3a, in order to compute $\mathbf{E}\{x(2)|y(1)\}$, we first take the Kalman filter estimate of the state at node 1 based on node 1 data, and then use the upward dynamics (6) to predict the value of the state at the parent node 4, and then use the downward dynamics (1) to obtain the estimate of the state at node 2 based on node 1 data. Likewise, as shown in Figure 3b, to obtain $\mathbf{E}\{x(3)|y(1), y(2)\}$, we merge the upward predictions of the state at node 4 based individually on nodes 1 and 2, and then predict downward to obtain the desired estimate at node 3.

Continuing, consider the whitening of nodes 5 – 8. Since all of these are to be whitened with respect to nodes 1 – 4, a common calculation has the information flow depicted in Figure 3c. That is, we take the Kalman filter estimate of the state at node 4 based on data from nodes 1 – 4, predict upward to node 13 and then downward to node 8. This estimate is then (a) predicted downward to node 5; (b) merged at node 8 with an upward predicted estimate from node 5 and then predicted downward to node 6; (c) merged at node 8 with the upward predicted and merged estimate based on nodes 5 and 6 and then predicted downward to node 7; and (d) merged at node 8 with the full predicted and merged estimate of the state at this node based on nodes 5 – 7 (i.e. the upward Kalman filter's predicted estimates at node 8). These results yield the estimates needed to compute $\nu(s)$ in at nodes 5 – 8. Similarly, as shown in Figure 3d, the Kalman filter estimates at nodes 4 (based on measurements at nodes 1 – 4) and 8 (based on measurements at nodes 5 – 8) are predicted upward, merged, and then predicted downward to node 12, providing a common piece of information used in an exactly analogous fashion to obtain the desired residuals at nodes 9 – 12.

Looking at the process we have just described, we see that it has similar structure to the

multiscale kalman filtering algorithm. The key differences are: (a) in the upward sweep we need to use *several* predicted estimates (e.g., at node 4 we make use of the prediction of the state at this node based on the measurement at node 1, measurements at nodes 1 and 2 together, and measurements at nodes 1 – 3); and (b) there is a downward sweep involving both pure downward prediction as well as merging of estimates in disjoint subtrees (e.g., in merging the estimate at node 8 based on the measurements at nodes 1 – 4 with that based on the measurement at node 5 before continuing the backward prediction to node 6). In effect, the computations required for whitening are a superset of those required for multiscale Kalman filtering, and as a result our algorithm has a multiscale Kalman filter embedded in it. We describe in detail below how the computations can be organized to obtain an efficient algorithm for likelihood calculation on q^{th}-order trees.

The required calculations in our algorithm can be broken up into three steps: an *upward sweep*, followed by a *downward sweep*, followed by the computation corresponding to (11). To describe the upward and downward sweeps, we begin by examining the structure of the set of data to be used in whitening $y(s)$ at some node s. In particular, this set of data consists of measurements at all descendant nodes of s, $Y_s^{\alpha_q}$, together with data at other nodes that have been placed earlier in the order. We define this latter set of nodes as $\bar{Y}_s = \{y(\sigma) | \sigma < s \text{ and } y(\sigma) \notin Y_s^{\alpha_q}\}$. Thus, the general equation for the residual $\nu(s)$ and its variance $\Lambda_{\nu(s)}$ is:

$$\nu(s) = y(s) - C(s)\hat{x}(s|\bar{Y}_s, Y_s^{\alpha_q}) \tag{12}$$

$$\Lambda_{\nu(s)} = C(s)P(s|\bar{Y}_s, Y_s^{\alpha_q})C^T(s) + R(s) \tag{13}$$

where $\hat{x}(s|\bar{Y}_s, Y_s^{\alpha_q}) = \mathbf{E}\{x(s)|\bar{Y}_s, Y_s^{\alpha_q}\} = \mathbf{E}\{x(s)|y(\sigma), \sigma < s\}$.

Since the sets $\bar{Y}_s$ and $Y_s^{\alpha_q}$ are disjoint, one way in which to compute the estimate $\hat{x}(s|\bar{Y}_s, Y_s^{\alpha_q})$ and the corresponding error covariance is to perform a merge operation on the estimates $\hat{x}(s|\bar{Y}_s)$ and $\hat{x}(s|Y_s^{\alpha_q})$ and their corresponding error covariances. Note that the latter of these estimates is *exactly the upward-predicted estimate calculated by the Kalman filter* (see (9), (10)). Also, note

that $Y_s^{\alpha_q}$ is nothing more than the union of the disjoint sets $Y_{s\alpha_1}, Y_{s\alpha_2}, \cdots, Y_{s\alpha_q}$ (e.g., the set of all descendants of node 8 in Figure 2 is the union of nodes 5, 6 and 7). We saw in the preceding section that we had need for estimates based on partial unions of these disjoint sets (e.g., we used estimates of node 8 based on node 5 alone, based on nodes 5 – 6, *and* based on nodes 5 – 7) and thus we define

$$Y_s^{\alpha_i} = \cup_{j=1}^{i} Y_{s\alpha_j}, \quad \text{for } i = 1, 2, \cdots, q \tag{14}$$

Examples of these sets are given in Figure 4 (for $s = \sigma$).

Using (14), we can identify the basic *downward recursive structure* of the sets $\bar{Y}_s$, which will provide us with a simple method for obtaining the estimates $\hat{x}(s|\bar{Y}_s)$ required in (12). In particular, if node s is the i^{th} of the immediate descendants of its parent node $s\bar{\gamma}$, i.e. if $s = s\bar{\gamma}\alpha_i$, then:

$$\bar{Y}_s = \bar{Y}_{s\bar{\gamma}} \cup Y_{s\bar{\gamma}}^{\alpha_{i-1}} \tag{15}$$

with the convention that $Y_{s\bar{\gamma}}^{\alpha_0} \equiv \emptyset$. An example of this is illustrated in Figure 4 in which we have indicated both the set of descendants of node s, $Y_s^{\alpha_q}$, and the three components $\bar{Y}_{s\bar{\gamma}}, \bar{Y}_{s\bar{\gamma}\alpha_1}$, and $\bar{Y}_{s\bar{\gamma}\alpha_2}$ of $\bar{Y}_s$, where the union of the latter two of these yields $Y_{s\bar{\gamma}\alpha_1} \cup Y_{s\bar{\gamma}\alpha_2} = Y_s^{\alpha_2}$. As we discuss in the detailed development below, both $\hat{x}(s|Y_s^{\alpha_q})$ and the set of estimates $\hat{x}(s|Y_s^{\alpha_i})$ for $i = 1, 2, \cdots, q-1$, are computed recursively through a series of *update-predict-merge* steps, during the upward sweep of the algorithm. In the downward sweep, we use the estimates $\hat{x}(s|Y_s^{\alpha_i})$ for $i = 1, 2, \cdots, q-1$ and the structure of $\bar{Y}_s$ characterized in (15) to recursively compute $\hat{x}(s|\bar{Y}_s)$ at each node, combine this with $\hat{x}(s|Y_s^{\alpha_q})$, and then use (13) to compute the residual $\nu(s)$.

We begin with the upward sweep of the algorithm where at each node s we wish to compute and store the set of predicted and partially merged estimates $\hat{x}(s|Y_s^{\alpha_i}), i = 1, 2, \cdots, q$. As in the upward sweep of the smoothing algorithm described in Section 2, we start by initializing the estimates $\hat{x}(s|Y_s^{\alpha_q})$ at the finest level of the tree to zero. Likewise, the covariances $P(s|Y_s^{\alpha_q})$

at the finest level are initialized using the Lyapunov equation (3). Suppose next that we have $\hat{x}(\sigma|Y_\sigma^{\alpha_1}), \hat{x}(\sigma|Y_\sigma^{\alpha_2}), \cdots, \hat{x}(\sigma|Y_\sigma^{\alpha_q})$ at each of the immediate descendants $\sigma = s\alpha_1, s\alpha_2, \cdots, s\alpha_q$ of node s. At each of these nodes it is only the last of these estimates, the original Kalman filter predicted estimate $\hat{x}(\sigma|Y_\sigma^{\alpha_q})$, that is used in the upward calculation to obtain the desired quantities at node s. First, $\hat{x}(\sigma|Y_\sigma^{\alpha_q})$ is updated to include the measurement at node σ using the Kalman filter update equations (4) – (5) (with s in (4) – (5) replaced by σ for $\sigma = s\alpha_1, s\alpha_2, \cdots, s\alpha_q$), yielding the updated estimate $\hat{x}(s\alpha_i|Y_{s\alpha_i})$, at each immediate descendant of s. These estimates are then predicted up the tree according to (7) – (8), yielding $\hat{x}(s|Y_{s\alpha_i}), i = 1, 2, \cdots, q$. What remains then is a horizontal recursion in which these q estimates are *successively* merged to form the desired estimates $\hat{x}(s|Y_s^{\alpha_i}), i = 1, 2, \cdots q$:

$$\begin{aligned}
\hat{x}(s|Y_s^{\alpha_i}) &= P(s|Y_s^{\alpha_i}) \sum_{j=1}^{i} P^{-1}(s|Y_{s\alpha_j}) \hat{x}(s|Y_{s\alpha_j}) \\
&= P(s|Y_s^{\alpha_i})[P^{-1}(s|Y_s^{\alpha_{i-1}}) \hat{x}(s|Y_s^{\alpha_{i-1}}) + P^{-1}(s|Y_{s\alpha_i}) \hat{x}(s|Y_{s\alpha_i})] \qquad (16) \\
P(s|Y_s^{\alpha_i}) &= [(1-i)P(s)^{-1} + \sum_{j=1}^{i} P^{-1}(s|Y_{s\alpha_j})]^{-1} \\
&= [P^{-1}(s|Y_s^{\alpha_{i-1}}) + P^{-1}(s|Y_{s\alpha_i}) - P(s)^{-1}]^{-1} \qquad (17)
\end{aligned}$$

Equations (16) – (17) for computing these merged estimates follow from the fact that the measurements in the sets $Y_{s\alpha_i}, i = 1, 2, \cdots, q$ are conditionally independent given $x(s)$. Thus we see that, as compared to the merge step (9) – (10) for the Kalman filter, the merge step for likelihood calculation involves a q-step horizontal recursion (16), (17) in which the last step yields the same quantity $\hat{x}(s|Y_s^{\alpha_q})$ as in the Kalman filter, but in which the preceding quantities $\hat{x}(s|Y_s^{\alpha_i}), i = 1, 2, \cdots, q$ are now explicitly computed and stored for later use in the downward recursion.

The upward update-predict-merge process continues up the tree until the root node is reached. At the end of the upward sweep, we have obtained at each node the set of estimates $\hat{x}(s|Y_s^{\alpha_i})$ for $i = 1, 2, \cdots, q$. The downward sweep then begins with the residual calculation at the root node, 0,

of the tree. Since $\bar{Y}_0 = \emptyset$, we have that:

$$\hat{x}(0|\bar{Y}_0, Y_0^{\alpha_i}) = \hat{x}(0|Y_0^{\alpha_i}), \quad i = 1, 2, \cdots, q \tag{18}$$

with a similar initialization for the error covariances $P(0|Y_0^{\alpha_i})$. The estimate and error covariance at $i = q$ are then used to calculate the residual and its covariance at the root node via (12), (13). Since the root node is the last one in our enumeration of nodes, the residual at this node is simply the Kalman filter residual. The other estimates in (18) for $i = 1, 2, \cdots, q-1$ provide initialization for recursive computation of $\hat{x}(s|\bar{Y}_s)$. In particular, using the recursive structure of the sets $\bar{Y}_s$ given by (15), we have that if $s = s\bar{\gamma}\alpha_i$:

$$\hat{x}(s|\bar{Y}_s) = A(s)\hat{x}(s\bar{\gamma}|\bar{Y}_{s\bar{\gamma}}, Y_{s\bar{\gamma}}^{\alpha_{i-1}}) \tag{19}$$

$$P(s|\bar{Y}_s) = A(s)P(s\bar{\gamma}|\bar{Y}_{s\bar{\gamma}}, Y_{s\bar{\gamma}}^{\alpha_{i-1}})A^T(s) + B(s)B^T(s) \tag{20}$$

Finally, these estimates are merged with the q estimates $\hat{x}(s|Y_s^{\alpha_i})$ computed during the upward sweep:

$$\hat{x}(s|\bar{Y}_s, Y_s^{\alpha_i}) = P(s|\bar{Y}_s, Y_s^{\alpha_i})[P^{-1}(s|Y_s^{\alpha_i})\hat{x}(s|Y_s^{\alpha_i}) + P^{-1}(s|\bar{Y}_s)\hat{x}(s|\bar{Y}_s)] \tag{21}$$

$$P(s|\bar{Y}_s, Y_s^{\alpha_i}) = [P^{-1}(s|Y_s^{\alpha_i}) + P^{-1}(s|\bar{Y}_s) - P(s)^{-1}]^{-1} \tag{22}$$

The estimate $\hat{x}(s|\bar{Y}_s, Y_s^{\alpha_q})$ is then used in the subsequent orthogonalization step given by (12) – (13), while the q estimates $\hat{x}(s|\bar{Y}_s)$, $\hat{x}(s|\bar{Y}_s, Y_s^{\alpha_i}), i = 1, 2, \cdots, q-1$ are propagated down the tree according to (19) – (20). At the end of the downward sweep we have obtained $\nu(s)$ and $\Lambda_{\nu(s)}$ at each node s, and (11) can then be used to compute the associated likelihood.

Both the upward and downward sweeps of the algorithm are parallelizable at each level of the tree. Consider first the upward sweep. The update step (4) at a given node s requires only the value of $\hat{x}(s|Y_s^{\alpha_q})$, which is available from computations at the previous level, and the measurement $y(s)$. Thus, all of the updates at any given level can be performed in parallel. Likewise, the prediction step (7) requires only the updated estimate, and the merge step (9) requires only the predicted

estimates $\hat{x}(s|Y_{s\alpha_j}), j = 1, 2, \cdots, q$. Similar statements can be made about the corresponding error covariance calculations and about the downward sweep computations in (12)–(13), (16)–(17), and (19) – (22).

Finally, we discuss the complexity of the algorithm as a function of the model parameterization and order of the tree. Recall that $x(s) \in \mathcal{R}^n, w(s) \in \mathcal{R}^m$ and $y(s) \in \mathcal{R}^p$. Using the approximation that inversion of an $N \times N$ symmetric matrix requires $N^3/3$ floating point operations (flops) it is straightforward to calculate the number of flops required by the algorithm (see Appendix A). In particular, if we assume that the measurements $y(s)$ are available at *all* nodes on the tree, then the algorithm requires[9] $25n^3/3 + 4p^3/3$ flops per node. Note that the total per-node computation is *constant.* Also, note that the structure of the model can often be exploited to substantially reduce the required computation, e.g., in the context of the multiscale model used for computing optical flow in [13], we could use the fact that the dynamics are diagonal. Likewise, in the texture discrimination application in the next section in which we use the approximate GMRF multiscale models proposed in [12], simplification results from the fact that the matrices $A(s)$ have large blocks of zeros, and the fact that measurements are only available at the finest level.

4 A Texture Discrimination Application

In this section we illustrate the use of the likelihood calculation algorithm in the context of texture discrimination. In the classical texture discrimination problem, we are given a set of texture models and a set of noisy random field observations, and we must choose that model which corresponds most closely in some sense to the data [9]. When statistical models are available for the textures and the measurements, this problem can be formulated as a likelihood ratio test and that is the

[9]The number here is approximate — for simplicity we have ignored quadratic, linear and cross terms in p and n. Obviously, such terms may be significant if n and p are small. A more detailed analysis in which these terms are accounted for can be found in Appendix A.

approach we take in this section. In particular, we will utilize GMRF models, and multiscale representations of these, as a basis for texture representation, and will examine for the discrimination problem the relative merits and performances of a GMRF-based LRT, a multiscale model-based LRT, and a minimum-distance classifier approach developed in [1]. Our analysis will be based on synthetic random field measurements which correspond to noisy realizations of GMRF texture models. We show that texture discrimination based on the multiscale methods described in this paper provides a computationally efficient approach that works in important contexts in which GMRF-based methods are either computationally infeasible or suffer significant losses in performance. In addition, we apply the methodology to discrimination of three textures contained in a set of synthetic aperture radar (SAR) imagery.

4.1 Gaussian Markov Random Fields and Their Multiscale Representations

Gaussian MRF's correspond, for some choice of parameters, to the following autoregressive model [2]:

$$z(i,j) = \sum_{(k,l)\in D} h_{k,l} z(i-k, j-l) + e(i,j) \tag{23}$$

where $h_{k,l} = h_{-k,-l}$, $(i,j) \in \{0,1,\cdots,M_1-1\}\times\{0,1,\cdots,M_2-1\}$, D is a neighborhood around $(0,0)$, and $e(i,j)$ is a locally correlated driving noise term. As in [1], we interpret the lattice as a *toroid*, i.e. the independent variables (i,j) in (23) are interpreted modulo (M_1, M_2). For instance, the first-order neighborhood of lattice site $(0,0)$ is given by the set $\{(0,1),(0,M_2-1),(1,0),(M_1-1,0)\}$.

In [12], we introduced a multiscale representation of (23) based on a generalization of the midpoint deflection construction of a Brownian motion over an interval. In addition, we also introduced in [12] a family of multiscale *approximate* GMRF representations, which allow one to trade off the complexity of the model (1), (2) for the accuracy in the representation. We provide the details of these representations in Appendix B. The fundamental result is a method for choosing the

multiscale model parameters such that (1) and (2) represent, to any desired degree of fidelity, the GMRF in (23).

4.2 The Texture Discrimination Problem

The noisy random field measurements on which the hypothesis test is based are of the form:

$$y(i,j) \;=\; c(i,j)z(i,j)+v(i,j) \quad 0 \le i \le M_1-1, 0 \le j \le M_2-1 \tag{24}$$

where $v(i,j) \sim \mathcal{N}(0, r(i,j))$, $c(i,j)$ is a spatially varying gain and $z(i,j)$ is a realization of a random field of the form (23). The spatially varying measurement gain $c(i,j)$ can be used to capture the possibility that measurements are available over only a subset $\mathcal{D}$ of the image lattice. In this case one simply sets $c(i,j) = 1, (i,j) \in \mathcal{D}$ and $c(i,j) = 0$ otherwise. We focus here on a binary hypothesis testing problem and denote the parameters of the multiscale models used in calculating likelihoods as[10] θ_0^{mm} and θ_1^{mm}, and the parameters of the corresponding GMRF models as θ_0^{gmrf} and θ_1^{gmrf}. As is well known, the likelihood ratio test (LRT) for deciding between two statistical models with parameters θ_0^{mm} and θ_1^{mm} and given by:

$$\log \frac{p_{y|\theta_1^{mm}}(Y|\theta_1^{mm})}{p_{y|\theta_0^{mm}}(Y|\theta_0^{mm})} \quad \begin{matrix} \text{Choose } \theta_1^{mm} \\ > \\ < \\ \text{Choose } \theta_0^{mm} \end{matrix} \quad \log \eta \tag{25}$$

results in optimum performance for the discrimination problem when (24) corresponds to measurements of a realization of a multiscale texture model. A similar test, based on θ_0^{gmrf} and θ_1^{gmrf} is, of course, optimal when the measurements correspond to a GMRF. We refer below to the LRT based on the GMRF and multiscale models as the GMRF-based and multiscale model (MM)-based approaches to texture classification, respectively. In the examples presented here, we have set $\eta = 1$,

[10] These parameters contain the information required to specify (1) and (2), i.e., $A(s), B(s), C(s), R(s)$ and P_0. In the context of representing GMRF's we discuss in [12] and Appendix B how the latter can be obtained.

corresponding to the maximum-likelihood decision rule.

To compare probability of error performance for the two approaches we have performed Monte Carlo simulations for a number of model pairs, image lattice sizes and noise levels. With regard to the choice of model pair, a number of GMRF models corresponding to natural textures are proposed in [3]. As observed there, two of these generate realizations which are quite similar visually. The parameters of these two models, which correspond to pigskin and sand, are given [3]. The specific results in this paper correspond to these two GMRF's, and in fact to the family of models given by $\theta_\omega^{gmrf} = (1-\omega)\theta_0^{gmrf} + \omega\theta_1^{gmrf}$, with $0 \leq \omega \leq 1$ (where θ_0^{gmrf} corresponds to pigskin and θ_1^{gmrf} to sand) and to a complementary family of multiscale approximate representations of the GMRF family constructed using the method developed in [12] and reviewed in Appendix B. The motivation behind choosing a family of models is that we want to illustrate that the performance characteristics of the MM and GMRF based approaches are comparable as the distance between the model choices varies. In particular, if we are trying to distinguish between observations coming from θ_ω and θ_1 for either MM or GMRF, this task is increasingly difficult as $\omega \to 1$ (at which point the two models are identical).

4.3 Complete Data

To demonstrate that the GMRF-based and multiscale model-based approaches to texture discrimination result in similar performance, we first compare their performance in the case that $r(i,j) \equiv r$, $c(i,j) \equiv c$, since in this case 2-D FFT's can be used to calculate the likelihoods required for the GMRF-based LRT. To implement the MM-based LRT, we need to make a choice of *which* multiscale models to use. We choose first and for most of our experiments the simplest of the multiscale approximate models corresponding to a given GMRF, namely a zeroth-order Haar-transform based model (see Appendix B). The state dimension n of this model is 16, and the measurement dimen-

sion at the finest scale is also 16 (in this application, the measurements are only available at the finest scale). Assuming that $M_1 = M_2 = M$ (see (23)), tedious but straightforward calculations show that in this case the algorithm of Section 3 requires approximately 6540 flops per-pixel and thus $6540M^2$ flops in total (see Appendix A). The likelihood corresponding to the GMRF model can be computed using 2D-FFT approaches in $\mathcal{O}(M^2 \log M)$ computations, which in this case leads to less computation than the MM-based approach for reasonable values of M.

The results of experiments in which we generated measurements according to (24) and then carried out three approaches to classification are given in Figure 5. In particular, in Figure 5a, each data point corresponds to 1000 experiments in which we generated a random field corresponding to θ_ω^{gmrf} or θ_1^{gmrf} (500 experiments each) for a 32×32 lattice and signal-to-noise ratio[11] of 0 dB, and then implemented the MM-based, GMRF-based and minimum-distance (MD)-classifier approaches to texture classification (the minimum-distance classifier we used is based directly on that in [1], which uses least-squares estimates of the parameters as a sufficient statistic). The percentages of correct classifications we have plotted are estimates of the probabilities of correct classification. We can characterize the error in these estimates by noting that if we define the sample mean as $\hat{p} = N_c/N$, where N_c is the number of correct classifications in N trials, then a simple application of the central limit theorem allows us to show, for example, that if $p = 0.5$, with $N = 1000$, and with 95% confidence, the error in $\hat{p}$ is less than 0.031, whereas if $p = 0.9$, the 95% confidence error in $\hat{p}$ is approximately 0.056.

Note that, as expected, as $\omega \to 1$ the percentage of correct classifications approaches 50 percent, reflecting the increasing similarity of the models. In Figure 5a, the GMRF-based approach is superior to the MM-based approach, since in the experiments the measurements actually did correspond to a GMRF. However, the difference in performance is small and of at best marginal or

[11] $\text{SNR} \equiv 10 \log c^2 \mathbf{E}\{z(i,j)^2\}/r$.

no statistical significance in view of the fact that, in any real application, the GMRF model is an idealization. In addition, both the GMRF and MM-based approaches significantly outperform the MD-classifier.

The results in Figure 5a are based on the simplest zeroth-order multiscale model. By increasing the order of the approximate models, the performance results will become progressively closer to one another. For instance, we have performed experiments using the first and second order approximate representations discussed in [12], for SNR = 0, and $M = 16$. The results of these experiments (10000 Monte-Carlo trials) are shown in Figure 5b. The improvement in performance with increasing model order is apparent. Numerous other examples which complement and further reinforce these results for a variety of cases are discussed in [11].

4.4 Incomplete Data

The results in the previous section and in [11] provide substantial evidence that the MM-based and GMRF-based approaches to texture classification provide comparable performance under a variety of conditions. In this section, the results of experiments are presented which provide further evidence of this and, more importantly, allow us to demonstrate how our multiscale framework can be used to calculate likelihoods given measurements over only a subset of the image lattice. This is one case in which our multiscale approach provides a potentially significant computational advantage over GMRF-based approaches.

Note that the measurement matrix $C(s)$ in (2) can vary as a function of node. In the approximate multiscale models, the values of the GMRF are represented as components of state vectors at the finest level of the tree, each value being represented in one state vector. Thus, setting $C(s) = I$ if s is a node at the finest level corresponds to the case of *complete* measurements, i.e. $c(i,j) = 1$ for all pairs (i,j). Likewise, when not all measurement data are available, we can take this into

account by eliminating the appropriate rows of the matrices $C(s)$. This is exactly what we have done in this section in which we used measurements over an incompletely sampled region as in Figure 6a.

Unavailable measurements correspond to black regions in Figure 6a, and might be single pixels or groups of pixels of various sizes. We have computed the relative performance of the GMRF-based and MM-based approaches on domains small enough to do the exact calculations for the GMRF models. Measurements of a GMRF random field were made at 90% of the 16×16 lattice sites, at SNR's of -10, 0 and 10 dB. The results are shown in Figure 6b and illustrate that the GMRF-based and MM-based approaches provide comparable performance. Again, in view of the fact that in any real application both of these models are idealizations, the performance differential is insignificant.

4.5 Application to Synthetic Aperture Radar Signal Processing

Finally, we present some results in the context of stripmap synthetic aperture radar (SAR) imagery [17]. Specifically, we have used the multiscale framework as a basis for discriminating between the three types of background clutter shown in Figure 7a: grass, trees, and road[12]. The data from which we generated this image were collected using a fully polarimetric SAR [19]. In this case, the processed radar return $Y(i,j)$ is a complex vector which consists of two co-polarization terms and one cross-polarization term:

$$Y(i,j) = \begin{bmatrix} HH \\ HV \\ VV \end{bmatrix}_{i,j} = \begin{bmatrix} HH_I + HH_Q \\ HV_I + HV_Q \\ VV_I + VV_Q \end{bmatrix}_{i,j} \tag{26}$$

[12]The SAR data were provided by the MIT Lincoln Laboratory under the ARPA sponsored Advanced Detection Technology program [19].

where HV_I and HV_Q are, for example, the in-phase and quadrature components of the vertically polarized return from the horizontally polarized transmit pulse. To generate Figure 7a we have processed the vector SAR image with the polarimetric whitening filter (PWF) developed in [18], which corresponds to setting:

$$y(i,j) \;=\; Y(i,j)^{\#}\hat{\Sigma}^{-1}Y(i,j) \tag{27}$$

where the # symbol denotes the complex conjugate transpose and $\hat{\Sigma}$ is the sample covariance matrix of the image. As discussed in [18], under certain conditions the PWF minimizes the speckle commonly associated with coherent imagery.

We interpreted the pixel values of the PWF image as measurements of a multiscale process as in (2) and used multiscale models for grass trees, and road as a basis for discriminating between clutter types. GMRF model parameter estimates were obtained from imagery taken nearby using sample correlation data as in [10, 8]. These were then transformed multiscale models using the methodology discussed in Appendix B. The image in Figure 7a was divided into 16 $\times$ 16 patches, and each patch was assigned classified according to which multiscale model (grass, trees, or road) had the highest likelihood. The result is shown in Figure 7b — white corresponds to road, light gray to trees and dark gray to grass. The classifications appear consistent with Figure 7a. Shadows in the lower and top left lead to classifications of some areas of trees as road. Likewise, bushes in the primarily grass upper right are classified as trees.

While Figure 7b is in essence a rudimentary segmentation of Figure 7a, we emphasize that this is not the goal. Rather, we view the ability of the multiscale framework to discriminate as indicative of the extent to which it can be used to *model* the underlying clutter. If models can be developed which capture the statistical structure of the clutter, than these can be used as a basis for the development of segmentation techniques far more powerfull than simple block discrimination (in analogy to methods such as those developed in [3, 16] subsequent to [1] for GMRF's) as well as

new approaches to such tasks as anomaly detection, change detection and target detection.

5 Conclusions

We have presented a likelihood calculation algorithm for a class of multiscale stochastic models. The algorithm exploits the structure of the tree on which the multiscale models are defined resulting in an efficient and parallelizable approach. In addition, we have investigated one possible application of the algorithm to the problem of texture discrimination, and have demonstrated that likelihood-based methods using our algorithm and the results in [12] for modeling GMRF's have substantially better probability of error characteristics than well-known minimum-distance classifier approaches, and achieve performance comparable to that of GMRF-based techniques, which in general are prohibitively complex computationally. Since our multiresolution algorithm has constant per-pixel complexity independent of data array size and does not require uniform sampling of the domain, it represents a very promising alternative to GMRF-based methods.

Acknowledgment: We thank John Henry, Bill Irving and Les Novak of the MIT Lincoln Laboratory for providing us with the SAR imagery and subsequent technical support.

A Complexity Analysis

In this appendix we analyze the computational complexity of the likelihood calculation algorithm presented in Section 3. Recall that the algorithm consists of an upward sweep (including update, predict and merge steps) and a downward sweep (including predict, merge and orthogonalize steps) followed by a step in which the likelihoods corresponding to the normalized residuals are added up. We will analyze each of these in turn. As in Section 2, we assume that the state dimension is n and the measurement dimension is p (the complexity is not a function of the driving noise

dimension). The number of branches in the tree is q. Finally, we assume that the computation of the process noise covariance (using the Lyapunov equation (3)) and the upward model parameters in (6) is negligible. This assumption is valid, for instance, when the multiscale model has parameters $A(s\alpha_i)$ and $B(s\alpha_i)$ which vary only as a function of scale or only as a function of scale and i, since in these cases the state covariance and upward model parameters vary similarly, and hence only need to be computed at one node at each scale (if the dependence is on scale only) or q nodes at each scale (if the dependence is on i as well). We recall that the computing the inverse of a symmetric matrix requires approximately $u^3/3$ floating point operations (flops), where u is the dimension of the matrix, that multiplying a $u \times v$ matrix by a $v \times w$ matrix requires approximately $2uvw$ flops, and that this latter computation can be reduced by approximately a factor of two if the matrices involved are symmetric.

Upward sweep update (4), (5): The computation of the gain $K(s)$ requires approximately $2n^2p + 4np^2 + p^3/3 + p^2/2$ flops. The update of the covariance given $K(s)$ requires an additional $2n^2p + n^3 + n$ flops. Finally, the update of the state estimate requires $4np + p + n$ flops. Thus, the total computation for an update step at each node requires approximately $n^3 + p^3/3 + 4n^2p + 4np^2 + 4np + p^2/2 + 2n + p$ flops.

Upward sweep prediction (7), (8): The prediction of the covariance requires approximately $3n^3 + n^2/2$ flops, whereas the prediction of the state estimate requires an additional $2n^2$ flops.

Upward sweep merge (16), (17): By merging recursively across the offspring of node s, the additional computation required to obtain $P(s|Y_s^{\alpha_i})$ is equal to $2n^3/3 + n^2$. The additional computation required to obtain the estimate $\hat{x}(s|Y_s^{\alpha_i})$ is equal to $4n^2 + n$ flops, for a total of $2n^3/3 + 5n^2 + n$ flops.

Downward sweep prediction (19), (20): Predicting the covariance requires $3n^3 + n^2/2$ flops whereas the complexity of predicting the estimate down is $2n^2$ flops.

Downward sweep merge (21), (22): Computing the covariance requires $2n^3/3 + n^2$ flops (recall that the inverse of $P(s|Y_s^{\alpha_i})$ has already been computed during the upward sweep). Computation corresponding to (21) requires an additional $4n^2 + n$ flops, for a total of $2n^3/3 + 5n^2 + n$ flops.

Orthogonalization (12), (13) and summation (11): The innovations covariance computation requires $2n^2p + 2np^2 + p^2/2$ flops, whereas computation of the innovation itself requires $2np + p$ flops. The summation requires $2p^3/3 + p^3/3 + 2p^2 + 2p$ flops per node, where we have assumed that the determinant computation requires $2p^3/3$ flops. Thus, the total per-node computation in these steps is $2n^2p + 2np^2 + p^3 + 5p^2/2 + 3p + 2np$ flops.

Adding these up, we calculate that the total complexity of the algorithm is, with $l + 1$ being the number of levels, $(q^{l+1}/(q-1))(25n^3/3 + 4p^3/3 + 6n^2p + 6np^2 + 15n^2 + 3p^2 + 6np + 4n + 4p)$ flops. Since the number of nodes is $q^{l+1}/(q-1)$, the total *per-node* computation is constant in the sense that, if the number of levels of the tree is changed, the per-node computation does not. With measurements only available at the finest scale, the total complexity of the algorithm is $q^l(n^3 + 4p^3/3 + 6n^2p + 6np^2 + 6np + 3p^2 + 2n + 4p) + (q^{l+1}/q - 1)(22n^3/3 + 10n^2 + 2n)$ flops, since the update and orthogonalize steps only need to be done at q^l nodes in that case, which again implies a constant per-node complexity (although in this case the per-node computation does depend on q).

Finally, we can use the above analysis to calculate the per-pixel computational complexity of the likelihood calculation algorithm as applied to the texture discrimination problems in Section 4. In this case, $n = p = 16$ (see Appendix B), $q = 4$, and measurements are only available at the finest level. An $M \times M$ image leads to a model with $M^2/16 + M^2/64 + \cdots + 1 \approx M^2/12$ nodes, and $M^2/16$ nodes at the finest level, and hence the total per-pixel computation in this case is approximately 6540 flops[13].

[13]This complexity could be further reduced by taking into account the special structure of the multiscale approximate GMRF models, e.g., as mentioned previously, the matrices $A(s)$ in these models have large blocks of zeros.

B Multiscale Representations of Markov Random Fields

In this appendix, we review the multiscale representations of Markov random fields introduced in [12]. These representations are based on a generalization of the classical mid-point deflection construction of Brownian motion. This construction is discussed first, followed by a review of our generalization to exact and approximate multiscale representations of MRF's.

B.1 Multiscale Representation of Brownian Motion

Our multiscale representations of 1-D Markov processes and 2-D MRF's rely on a generalization of the mid-point deflection technique for constructing a Brownian motion in one dimension. To construct a Brownian motion sample path $b(t)$ over the interval $[0, 1]$ by mid-point deflection, we start by randomly choosing values for the process at the two boundary points and at the mid-point, i.e. we choose the three numbers $[b(0), b(0.5), b(1)]$ according to the joint probability distribution implied by the Brownian motion model. We then use these three values to predict values of the Brownian motion at the one-fourth and three-fourths points of the interval – the Bayesian estimate of the mid-points just corresponds to linear interpolation as shown in Figure 8(a). Random values, with appropriate error variances, are then added to the predictions at each of these new points, as seen in (b). The critical observation to be made here is that, since the Brownian motion process is a Markov process, its value at the one-fourth point, given the values at the initial point and mid-point is *independent* of the process values beyond the mid-point, in particular the values at the three-fourths and end-points of the interval. Obviously, it is also the case that the value at the three-fourths point is independent of the values at the initial and one-fourth points, given the values at the mid-point and final point. What this means for us is that this construction can be interpreted as a sample realization of a particular multiscale model. This model has as its root-node state the 3-tuple $[b(0), b(0.5), b(1)]$, and two states at the second level given by $[b(0), b(0.25), b(0.5)]$

and $[b(0.5), b(0.75), b(1)]$, as shown in Figure 8(d). The Markov property of Brownian motion allows us to iterate the mid-point deflection construction and its equivalent multiscale model, generating values at increasingly dense sets of dyadic points in the interval. At each level in this procedure we generate values at the mid-points of all neighboring pairs of points. In fact, since the only properties of the Brownian motion that we have used are its Gaussianity and Markovianity, this approach can be generalized to represent all 1-D Gauss-Markov processes within the multiscale framework [12]. Further, by generalizing the multiscale model class appropriately, *all* 1-D Markov processes can be represented [12].

B.2 Exact Multiscale Representations of GMRF's

The representations of 1-D Markov processes in the previous section relied on the conditional independence of regions inside and outside a boundary set, and we use the same idea here to represent Markov random fields on a square lattice. The multiscale model is identical to that used in the 1-D case, except that it is defined on a *quadtree* instead of a dyadic tree. That is, we consider multiscale models *exactly* as in (1) but where s denotes a node on a quadtree.

Consider a 2-D GMRF $z(t)$ defined on a $2^N \times 2^N$ lattice. The construction of Markov processes in 1-D started with the values of the process at the initial, middle and end points of an interval. In two dimensions, the analogous top level description consists of the values of the GMRF around the outer boundary of the lattice and along the vertical and horizontal "mid-lines" which divide the lattice into four quadrants[14]. For instance, on a 16×16 lattice, the state vector x_0 at the root node of the quadtree contains the values of the GMRF at the shaded boundary and mid-line points shown in Figure 9a. To construct a sample path of the GMRF, we begin by choosing a sample

[14]Strictly speaking, two mid-lines are not required. However, we take this approach here since it leads much more naturally to the approximate representations of GMRFs which are discussed in the next subsection and which form the basis for our experiments in Section 4.

from the joint pdf of the GMRF values defined on the boundary and mid-line set.

In the 1-D case, transitions from the first to second level consisted of obtaining a sample from the conditional distribution of the state at the mid-points of the left and right half-intervals. In two dimensions, we predict the set of values at the mid-lines in each of the four quadrants. The components of the four state vectors at the second level are illustrated in Figure 9b for the 16×16 GMRF. Now, we can iterate the construction by defining the states at successive levels to be the values of the GMRF at boundary and mid-line points of successively smaller subregions. Because of the Markov property, at each level the states are conditionally independent, given their parent state at the next higher level. Thus, the GMRF can be thought of precisely as a multiscale stochastic process and this leads to models *exactly* as in (1).

B.3 Approximate Multiscale Representations of GMRF's

In this section we describe a family of approximate representations for Gaussian MRF's that provide low-dimensional alternatives to the exact multiscale representations. The basic idea behind the approximate representations is to take as the state not boundaries of regions, but rather some reduced-order representation of them. Conceptually, we would like to retain only those components of the boundary that are required to maintain nearly complete conditional independence of regions.

As described in detail in [12], one way to do this is to view the GMRF values which make up a particular state of the multiscale model as a set of 1-D sequences. For instance, consider the values of the GMRF contained in the root-node state (see Figure 9a), and in particular the values denoted with $\triangledown, \triangleleft, \triangleright, \triangle$ which make up the boundary of the north-west quadrant. We can view these as a set of four 1-D sequences, each of which has a length that is half the number of rows or columns in the lattice. Any given sequence is just as well represented in the wavelet transform domain [7, 15], and there are good reasons to believe that only a small number of wavelet coefficients are required

to represent essentially all of the information in a given sequence [12]. This suggests transforming the state via a wavelet transform, and only keeping a subset of the coefficients as our representation of the state. In the simplest representation, we could retain just the *averages* of the various 1-D sequences, which results in a substantial reduction in the dimensionality of the multiscale model. By keeping more coefficients in the expansion, we obtain a better representation of the original GMRF, and in the extreme case that all wavelet coefficients are kept, the representation is, of course, exact. Hence, this provides a flexible framework allowing one to tradeoff representation complexity and fidelity, and our main goal is to provide representations which have a complexity low enough to allow for substantial computational advantages within the multiscale framework, while also providing equal or better performance.

In our experiments in Section 4, we have used the simplest possible approximate representation, i.e. that which retains only the average of each 1-D sequence as the state. We refer to this as a zeroth-order Haar-transform based model since only the scaling coefficient in a Haar transform representation of the sequence is retained (for generalizations to higher order representations and other wavelets we refer the reader to [12]). Since, referring to Figure 9a, there are sixteen boundary sequences, the state dimension in this model at the coarsest level is sixteen. The state dimension is also sixteen at the next level (see Figure 9b). If we proceeded to the next level each of the boundary and mid-line sequences would be of length two. At this point, the region in the image corresponding to these boundary and mid-line sequence is 4×4. However, the values do not completely represent the state since the "basis" in this case — i.e., the two-pixel averages — is not complete. Hence, at this level, we simply take as the state the values of the sixteen pixels. At this point, all pixels in the image are represented and no more levels need to be added to the model. Thus, the state dimension is sixteen at all levels and in fact at all nodes. An $M \times M$ image thus leads to a quadtree multiscale model with $M^2/16$ nodes at the finest level. Since in an l-level model there are 4^{l-1}

finest level nodes, the number of levels in a multiscale model corresponding to an $M \times M$ image (where M is a power of 2) is $l = (\log_2 M) - 1$.

References

[1] R. Chellappa and S. Chatterjee. "Classification of textures using Gaussian Markov random fields". *IEEE Transactions on Acoustics, Speech and Signal Processing*, 33:959–963, 1985.

[2] R. Chellappa and R. Kashyap. "Digital image restoration using spatial interaction models". *IEEE Transactions on Acoustics, Speech and Signal Processing*, 30:461–472, 1982.

[3] R. Chellappa, B. Manjunath, and T. Simchony. "Texture segmentation with neural networks". In *Neural Networks for Signal Processing.* Prentice-Hall, 1992.

[4] K. Chou, A. Willsky, and A. Benveniste. "Multiscale recursive estimation, data fusion and regularization". Technical Report LIDS-P-2085, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, 1991. To appear in *IEEE Transactions on Automatic Control*, April 1994.

[5] K. Chou, A. Willsky, A. Benveniste, and M. Basseville. "Recursive and iterative estimation algorithms for multiresolution stochastic processes". In *Proceedings of the IEEE Conference on Decision and Control*, 1989.

[6] F. Cohen, Z. Fan, and M. Patel. "Classification of rotated and scaled textured images using Gaussian Markov random field models". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:192–202, 1991.

[7] I. Daubechies. "Orthonormal bases of compactly supported wavelets". *Communications on Pure and Applied Mathematics*, 91:909–996, 1988.

[8] H. Derin, P. Kelly, G. Vezina, and S. Labitt. Modeling and segmentation of speckled images using complex data. *IEEE Transactions on Geoscience and Remote Sensing*, 28:76–87, 1990.

[9] R. Kashyap, R. Chellappa, and A. Khotanzad. "Texture classification using features derived from random field models". *Pattern Recognition Letters*, 1:43–50, 1982.

[10] P. Kelly, H. Derin, and K. Hartt. Adaptive segmentation of speckled images using a hierarchical random field model. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36:1628–1641, 1988.

[11] M. Luettgen. *Image Processing with Multiscale Stochastic Models.* PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, May 1993.

[12] M. Luettgen, W. Karl, A. Willsky, and R. Tenney. "Multiscale representations of Markov random fields". *IEEE Transactions on Signal Processing*, 41:3377–3396, 1993.

[13] M. Luettgen and A. Willsky. "Efficient multiscale regularization with applications to the computation of optical flow". *IEEE Transactions on Image Processing*, 3:41–64, 1994.

[14] M. Luettgen and A. Willsky. "Multiscale smoothing error models". Technical Report LIDS-P-2234, Massachusetts Institute of Technology, Laboratory for Information and Decision Systems, 1994. To appear in *IEEE Transactions on Automatic Control.*

[15] S. Mallat. "Multi-frequency channel decomposition of images and wavelet models". *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:2091–2110, 1989.

[16] B. S. Manjunath and R. Chellappa. "Unsupervised texture segmentation using Markov random field models". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:478–482, 1991.

[17] D. Munson and R. Visentin. A signal processing view of strip-mapping synthetic aperture radar. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37:2131–2147, 1989.

[18] L. Novak and M. Burl. Optimal speckle reduction in polarimetric SAR imagery. *IEEE Transactions on Aerospace and Electronic Systems*, 26:293–305, 1990.

[19] L. Novak, M. Burl, R. Chaney, and G. Owirka. Optimal processing of polarimetric synthetic aperture radar imagery. *The Lincoln Laboratory Journal*, 3:273–290, 1990.

List of Figures

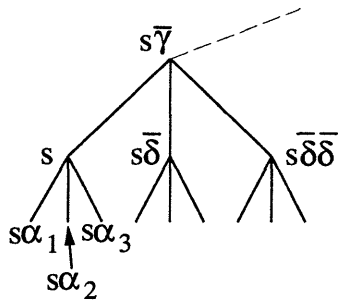


Figure 1

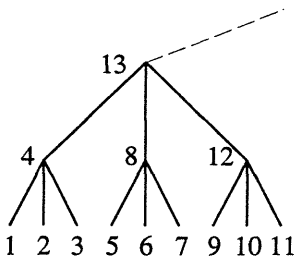


Figure 2

Figure 1: A third-order tree.

Figure 2: An ordering of the nodes on the tree which leads to efficient likelihood calculation.

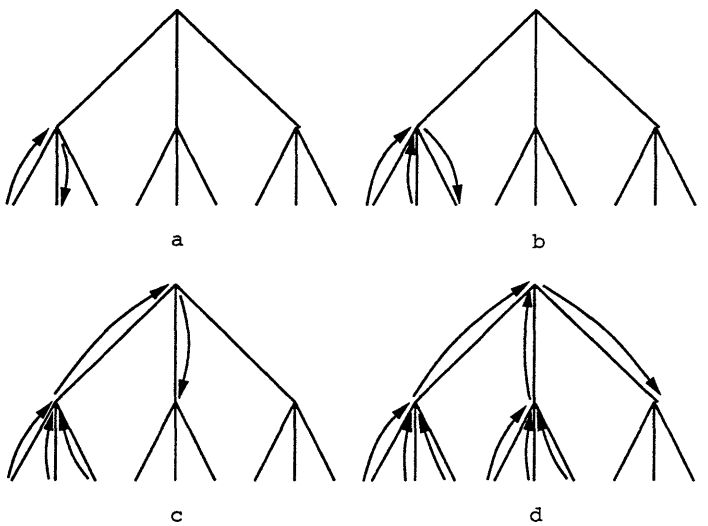


Figure 3: Information flow in the likelihood calculation algorithm.

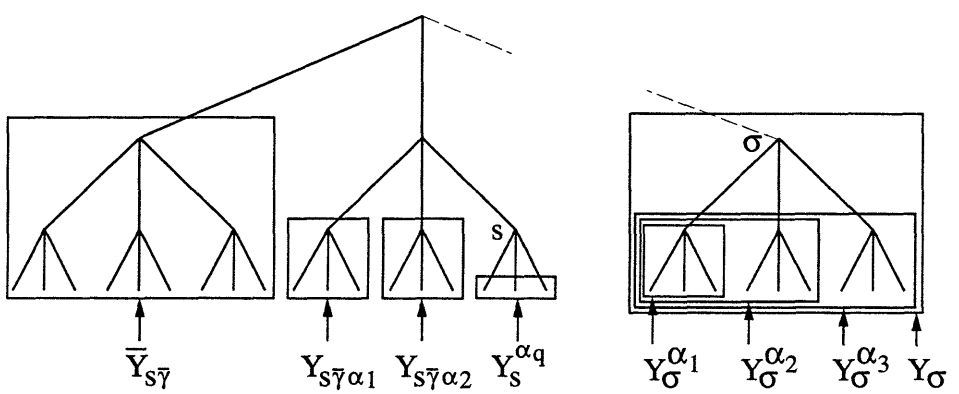
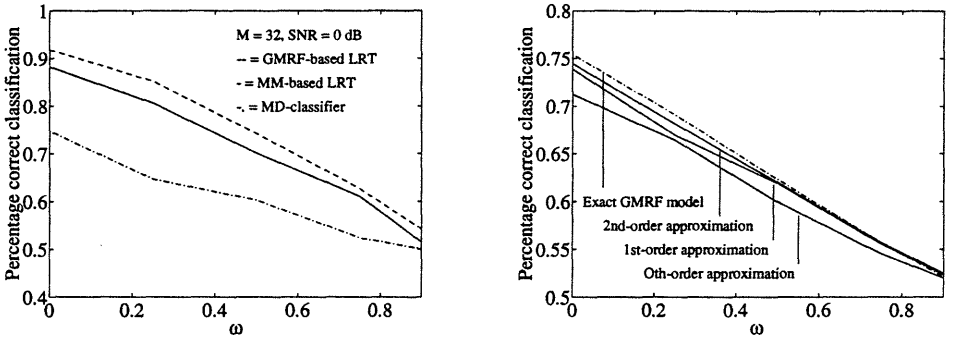
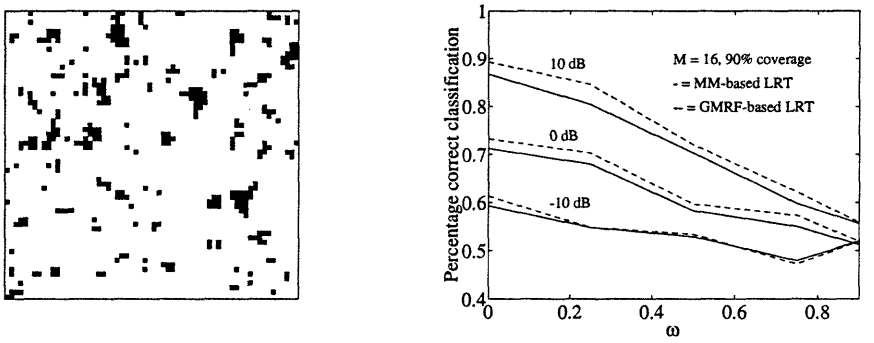


Figure 4: Examples of subsets $Y_s^{\alpha_i}$ and $\bar{Y}_s$ of nodes defined with respect to node s.



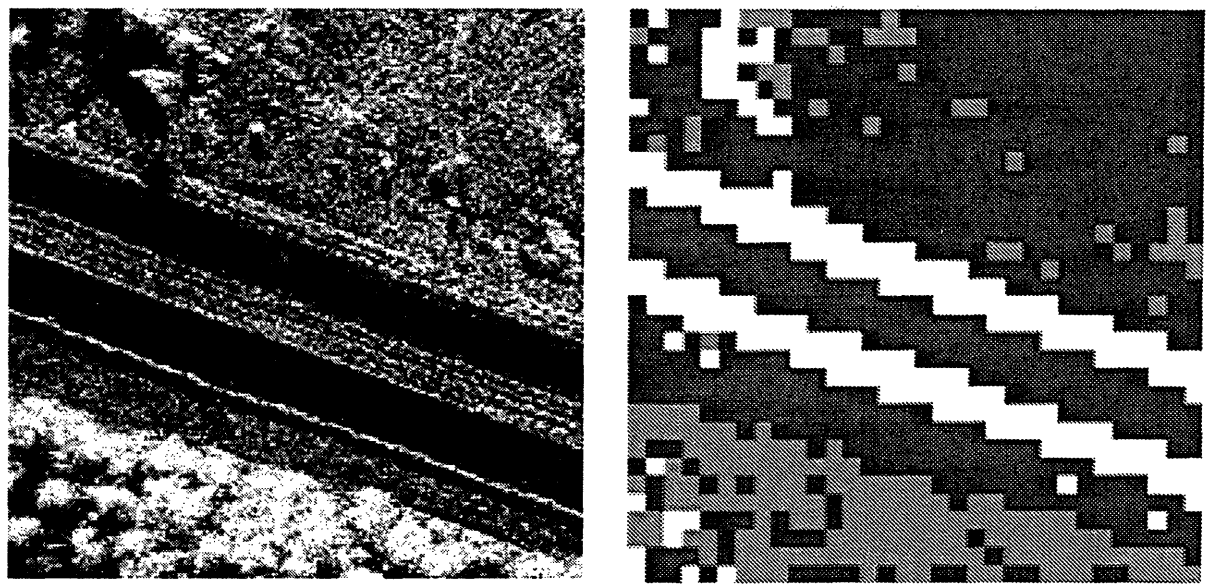
a b

Figure 5: (a) Comparison of multiresolution model (MM) based, GMRF-based and minimum-distance (MD) classifier approaches to texture classification. (b) Improvement in performance as the approximate GMRF representation order is increased.



a b

Figure 6: We demonstrate in Section 4.4 how the multiscale framework can be used to calculate likelihoods when the data is sampled in an irregular fashion. For instance, data may not be available the blackened regions in (a) due to dropouts, camera blockage or sampling constraints. (b) Performance data corresponding to 90% coverage.



(a) (b)

Figure 7: (a) Synthetic aperture radar image with three textures: grass, road and trees. (b) Classification of 16 by 16 patches.

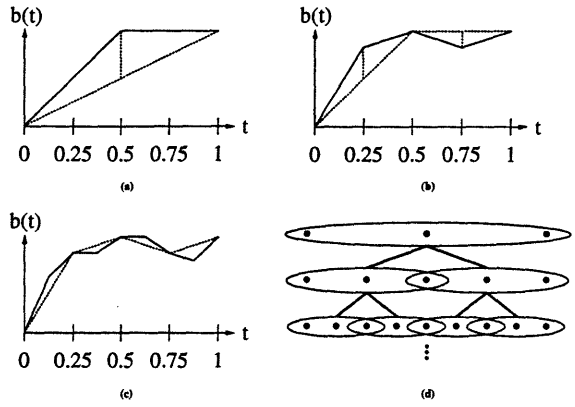


Figure 8: Brownian motion representation with a multiscale model.

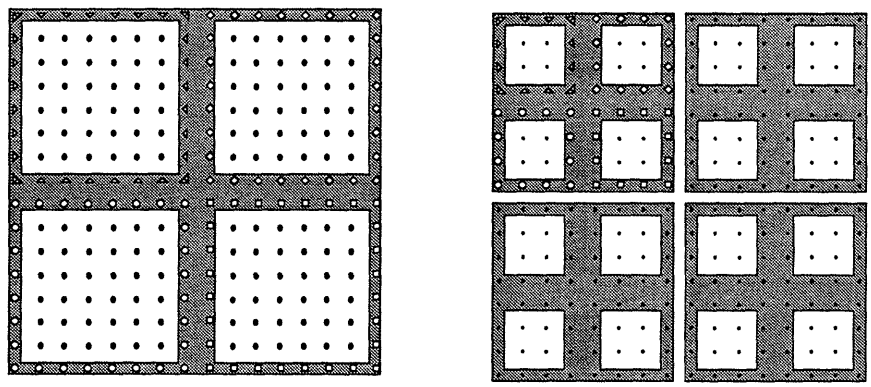


Figure 9: (a) The state at the root node in an exact multiscale representation and (b) the four states at the second level of the quadtree.