AUTOMATIC DETECTION OF EXPLOSIVE DEVICES IN INFRARED IMAGERY USING TEXTURE WITH ADAPTIVE BACKGROUND MIXTURE MODELS

A Thesis
presented to
the Faculty of the Graduate School
at the University of Missouri-Columbia

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

---

by
CHRISTOPHER SPAIN
Dr. James Keller, Thesis Supervisor
MAY 2011

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

AUTOMATIC DETECTION OF EXPLOSIVE DEVICES IN
INFRARED IMAGERY USING TEXTURE WITH
ADAPTIVE BACKGROUND MIXTURE MODELS

presented by Christopher J. Spain,

a candidate for the degree of Master of Science,

and hereby certify that, in their opinion, it is worthy of acceptance.

____
Dr. James Keller, Ph.D.

____
Dr. Dominic Ho, Ph.D.

____
Dr. Mihail Popescu, Ph.D.

ACKNOWLEDGEMENTS

I would like to thank Dr. James Keller for being my advisor, mentor and teacher, and for giving me a position as a research assistant. I appreciate his advice, time, and patience during the preparation of this thesis. Thanks are due to Dr. Dominic Ho for agreeing to be on my committee and for everything I have learned from him both in his class and outside. I am grateful to Dr. Mihail Popescu for being on my committee and for his help over the course of the past two years. Dr. Derek Anderson deserves thanks for his time, advice and help on understanding both the process of writing, and the subject matter of this thesis. Finally, I would like to thank Dr. Timothy Havens for getting me involved in research in the first place, mentoring me as an undergraduate and introducing me to the art of MATLAB programming.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

Introduction

1.1 Overview of problem

The detection of buried explosive hazards has long been a problem that concerns both civilians and the military. The need for methods to aid in detection of these objects has intensified in recent years and much research has gone into finding and evaluating solutions. One promising solution is to combine multiple sensors to find evidence of the presence of buried objects. One of the sensors that shows potential is the Long Wave Infrared (LWIR) camera. Not only are sensors being used to aid in the detection of explosive hazards, but there is a great need for some automated way of collecting information from the different modalities and for passing this information to a human user in a clear, understandable way. Computer techniques such as automated target recognition and queuing allow users even with limited experience in interpreting IR or radar to effectively use these sensors to find hazards.

Proposed here is a method of extracting areas of interest in LWIR imagery in the form of image chips which are then passed to a texture-based analysis and classification system. The evidence from both sources is combined and a determination made as to whether or not something is a target. The texture based analysis evidence can be used in multiple ways. This thesis will examine two possible combinations; the first is to use a background modeling technique to screen for potential hazards in the IR, then use texture analysis to reduce the false alarms generated by the pre-screener. The second proposed use is to combine the evidence from both systems and output a confidence as to whether an object is a hazardous anomaly. As this is a multi-component system, care has to be taken that each part is functioning properly and parameters are adjusted for best results. For this reason each component is verified on a simple case before being applied to the IR. Finally all components are combined to form the finished detector.

First, a background modeling technique using Gaussian Mixture Models (GMM) will be examined and tested on a simple case of a static background with easily differentiable foreground objects. Next, the GMM will be applied to the IR imagery and evaluated as if it were a standalone detector. The GMM was originally conceived for use with static cameras, so the application to video taken from a moving camera is a departure from its common usage.

The second step is to build the texture analyzer which will be tested on some simple, clear textures taken from a database to prove the concept before it is then applied it to the infrared imagery. Although many of the techniques used in the texture analyzer are widely known, this will be a unique combination and application of these techniques and will provide a potential means of automatic detection of explosive hazards in infrared imagery. It is also hoped that something will be learned about the texture analysis measures themselves as well as the nature of texture in digital imagery. Once evaluation of the two main techniques is complete they will be combined in the expectation that the performance of the dual system will better than that of either the GMM or texture analysis alone.

Chapter 2

Background & Related Work

This section will focus first on detailing the theory behind the Gaussian mixture models as well as briefly examining alternative background modeling techniques. Various texture analysis techniques and theory will then be examined. Before discussing the detection techniques it is important to give some background about the data being used and the theory behind detecting buried objects in infrared imagery.

2.1 Infrared Theory and Imagery

The imagery studied in this thesis is generated by a detector that operates in the Long-Wave Infrared (LWIR) band of the electromagnetic spectrum. This band roughly corresponds to wavelengths between 8 µm to 15 µm with the exact range depending on which source you read. The sensor used to generate the data here was has a sensitivity across in the range of 8 µm to 12 µm.

LWIR is detected using a sensor known as a microbolometer. The bolometer was invented as far back as 1878 by Samuel Pierpont Langley and originally consisted of two metallic strips coated in an absorbing material; in the case of the first bolometer this was a type of black deposit collected from oil lamps [1]. The idea is that the material absorbs the electromagnetic radiation then increases its temperature in proportion to the amount of energy absorbed. This temperature change can then be detected directly or indirectly. Commonly, the absorbent material coats a metallic element which then changes its resistance according to the temperature of the surface. Even very small changes in resistance can be detected by a simple circuit such as a Wheatstone bridge [1]. Modern microbolometers use an array of detecting circuits on a silicon substrate. The absorbing material is often either amorphous silicon or vanadium oxide. The absorbent normally sits around 2 μm above the background substrate. This keeps the absorbent thermally isolated from the substrate and also allows a reflecting material to be placed underneath to help absorb any energy passing through [2]. The changes in electrical resistance at each cell in the array is measured and converted into an electrical signal then used by the camera to form the image [2]. Figure 2.1 below shows a simplified diagram of the microbolometer components.

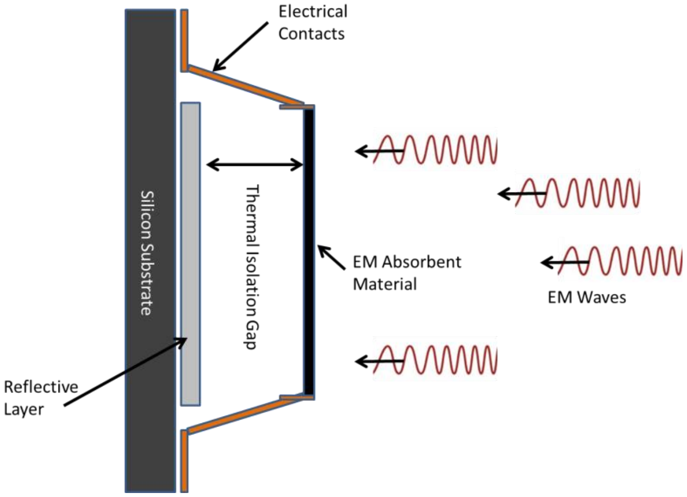


Figure 2.1. Simple diagram of a microbolometer showing main components.

2.1.1 Detecting Buried Objects with LWIR Cameras

In the infrared imagery there is often an area of high contrast, either a bright spot or a dark spot against the background surface (see Figure 2.2). These spots may correspond with locations of buried objects. The difference in appearance of these locations is due to thermal differences in the soil surface surrounding and above a buried object compared to the normal road surface. The presence of an object beneath the ground changes the heat capacity, thermal diffusivity and conductivity of that location, and therefore the amount of IR emitted is also changed [3]. Research into thermal properties of soil and buried explosive hazards have found that the presence of a large buried object reduces the thermal conductivity of that region of ground and so heat cannot flow or disperse as easily as it normally does. The effect of this during the day is that sunlight warms the ground causing the soil above the buried objects to be hotter than surrounding regions because the heat cannot flow as easily through the object as it can the soil [3]. It has also been found that at night and early morning the ground above buried objects is cooler (shows darker in the IR), during the day the area is hotter (shows white in IR) and the biggest difference is seen around local noon [3]. This matches observations in the LWIR studied in this thesis.

The theory that the presence of buried objects has an effect on the thermal properties of that location means that the signature for the objects depends heavily on the size and burial depth of the objects. If an object is small or buried deeply then its effect on the heat flux through the ground is likely to be quite small making it hard to detect.

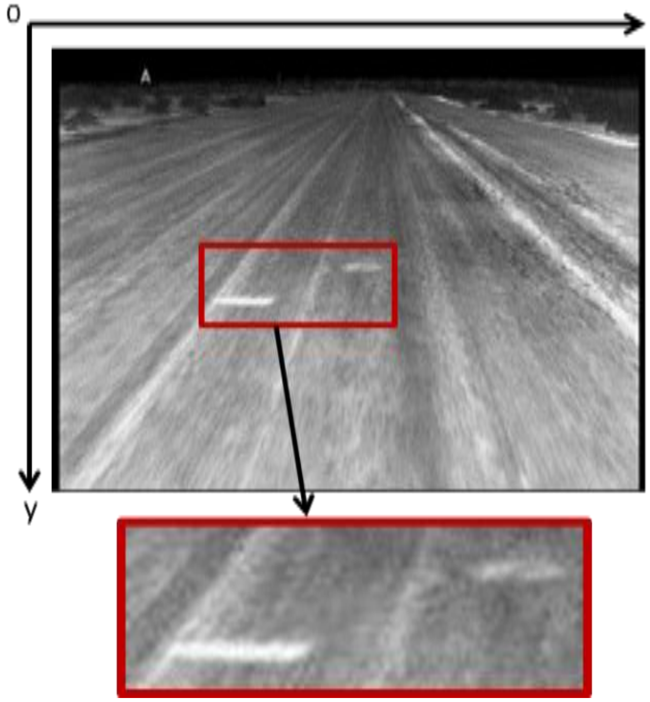


Figure 2.2. Sample image from the IR camera showing bright regions corresponding to known buried objects. The coordinate system seen here with the origin at the top-left will be used throughout.

The data used in this thesis was provided by the US Army Night Vision & Electronic Sensors Directorate (NVESD). The sensor is an infrared camera operating in the long-wave band and mounted on a Humvee (Figure 2.3). The system contains multiple cameras. In our case, we use data from the wide field of view infrared camera which delivers 8-bit, 640x480 image frames. It should be noted that 640x480 is considered good resolution for an infrared camera and 1024x768 microbolometers appear to be expensive and somewhat rare. All IR images in this thesis use the common image processing coordinate system used in Figure 2.2 where the origin is at the top-left of the image.

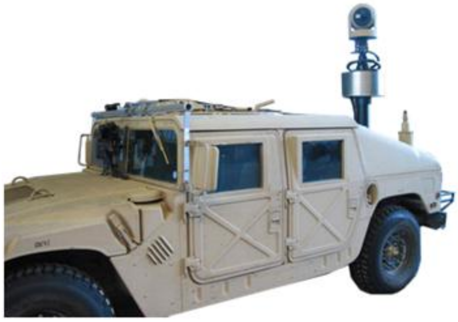


Figure 2.3. US Army NVESD test vehicle showing various mounted cameras.

2.2 Background Modeling

A number of methods have been used to model background in video. These techniques are commonly used to detect and track moving objects such as cars in traffic video [4], people moving in a room [5] and even piglets in a pen [6]. Here, a few of these techniques will be highlighted, a more comprehensive review is available in [4]. It might be useful to point out that what exactly constitutes background varies depending on the user and the application for which it is being used. In general though, background is normally defined as anything that is not foreground. Foreground is usually some group of objects of interest in a video image segment that may move at a different (faster) rate than other objects or have other unique characteristics. In the case of a camera mounted on a vehicle, everything is moving, but because parts of the image are fairly uniform such as the road, then the apparent motion is reduced and objects on the road appear as though they are approaching the camera.

2.2.1 Frame Differencing

In the simple case where there is a fixed camera and the foreground objects move relative to the camera, it is possible to separate the foreground from the background by simply subtracting the previous frame from the current frame and then determining if the difference is greater than some threshold. This technique is known as frame differencing [7]. This subtraction removes image areas that remain the same while exposing those that are different. Any part of the image that has changed between frames will have a non-zero value while any pixels that remain the same will have a value of zero. If the resulting pixel has an absolute value greater than some threshold, this can called as foreground.

$$FG(x, y) = \begin{cases} 1, & |I_t(x, y) - I_{t-1}(x, y)| > T \\ 0, & else \end{cases} \tag{2.1}$$

Where $I_t(x, y)$ is the value of the pixel located at (x, y)at time *t* and T is a fixed threshold. The process is aggregated over a number of frames, and those pixels where $FG(x, y) = 0$ are regarded as background. While this technique is simple, and often works well for a static camera, it does not work well for cameras affixed to moving vehicles as there is change in almost every pixel value as objects in the background appear to move toward the camera.

2.2.2 Median Filtering

Another simple method for background modeling involves calculating the median of all the values of a pixel over all currently observed frames and using the median value as the background model. A test pixel is classified as foreground if its difference from the median is larger than a certain threshold. This is rather inefficient as it requires buffering all the frames seen so far causing the amount of data stored to grow rapidly. For example, if there is imagery with a frame size of 800x600 pixels, 8-bit grayscale values and a frame rate of 15 frames a second, then after only 15 minutes there will be more than 6 GB of data to store.

Although more efficient implementations have been developed such as approximating the median using counters [4], this technique suffers from many of the same issues as frame differencing when it comes to moving cameras.

2.2.3 Gaussian Mixture Models

Gaussian Mixture Models (GMM) have been used successfully in subtracting background from video sequences taken with static cameras. For example, Stauffer and Grimson [8] used GMM background subtraction to aid in the tracking of objects such as vehicles in scene taken from a live roof-mounted camera. The basic idea is to treat each pixel in the video frame as a “pixel process” which changes over time as the frame sequence advances. The GMM approach models each pixel as a mixture of Gaussian distributions that gets updated online to reflect the pixel values encountered. After a short training period, say a few tens of frames, the model should converge to represent the most commonly seen pixel values. Any new pixel value encountered that does not fit within a certain distance of this model (i.e. the pixel value is far from the mean values of any of the Gaussian components) is determined to be the foreground. If a new pixel value matches the model, then it is considered as the background and the model is updated to reflect the new value.

Updating the GMM model involves moving the means of the Gaussian components towards the new matched pixel values. The standard deviation of each component is also updated following the mean update.

If a new pixel value does not match with any of the mixture components, but it occurs frequently, then this value will eventually be incorporated into the model. This suggests that the system is constantly adapting to the input sequence. Not only can the GMM account for multi-modal distributions of pixels in the short term, such as those introduced by moving leaves, a flag blowing, or passing shadows, but it can also account for longer-term changes such as changes in illumination between night and day, and change in weather conditions.

Of the methods surveyed, the one chosen to model the background in the IR, is the Gaussian Mixture Model outlined above. The GMM will now be described in more detail.

Following [8], the intuition behind this technique can be arrived at by storing all values of a given pixel over all frames in a video sequence, then plotting the histogram of these values. Often it is the case that a normal curve can be fitted to the histogram and the distribution through time estimated as Gaussian with a given mean and standard deviation (Figure 2.4). The image sequence can now be re-run with this Gaussian as the background model for that pixel. By comparing the pixel's value with the model on a per-frame basis, it is possible set some measure where, if a pixel value is close to the model mean it is called as background, else it is determined to be foreground. One possible formalization of this idea is to use a function that outputs binary values.

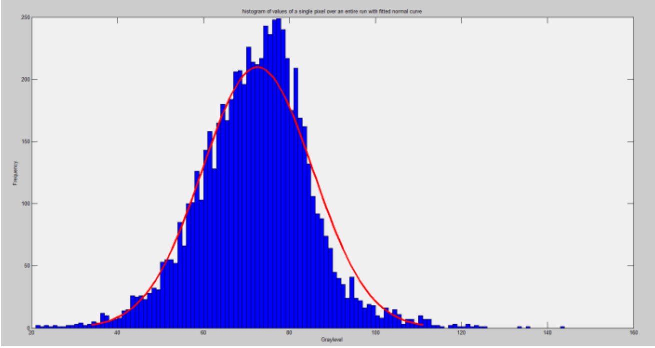


Figure 2.4. Histogram of pixel values for a single pixel location over the entire run. In this example, we can model the distribution using a single Gaussian function.

Define $FG(x,y)$ as a binary function that identifies foreground pixels as follows

$$FG(x,y) = \begin{cases} 1, & |I_t(x,y) - \mu(x,y)| > T \\ 0, & else \end{cases} \tag{2.2}$$

where $I_t(x,y)$ is the value of a pixel at location (x,y) at time (frame) t, $\mu(x,y)$ is the mean of Gaussian background model for pixel location (x,y), and T is a user-defined distance threshold.

There are several potential issues with this technique: the first is how to initialize the parameters, and second, the distribution may be multimodal. It is often the case that a pixel in a particular location can take on multiple legitimate yet very different background values. For example, a pixel near a tree that has a leaf blowing back and forth in the wind may have a multi-modal distribution as in Figure 2.5.

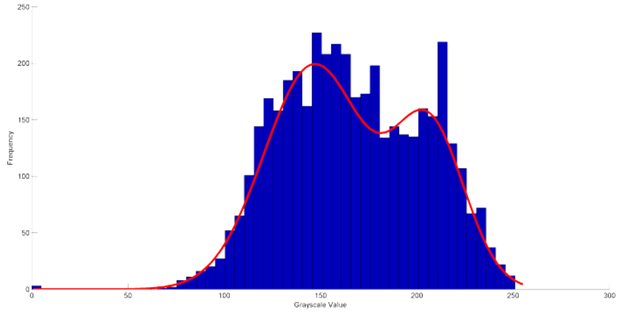


Figure 2.5. Histogram of pixel values for a single pixel location over the entire run. In this example, the pixel values show a bimodal distribution which can be modeled using a mixture of two Gaussian functions. In this case the pixel value alternates between the road and horizon depending on the motion of the vehicle.

The multimodal distribution of the background can be modeled using multiple Gaussian components; each one representing a subset of the major pixel values. The first problem is solved by allowing the model to adapt and change through time. Instead of an a posteriori model built on all frames from some training run, a base model is initialized then adapted as new frames are encountered. This way the parameters can be learned online and there is no need to decide a priori how the background model should appear. This adaptive process allows for imagery that changes drastically during a run, say from day to night, etc. while remaining accurate (after some period of re-adaptation). Also it obviates the need for a separate training run.

The probability $P(X_t)$ at time *t* that a particular pixel belongs to the background model is given by the Gaussian mixture model taken from [8] and is in the form of:

$$P(X_t) = \sum_{i=1}^{K} \omega_{i,t} \cdot \eta(X_t, \mu_{i,t}, \Sigma_{i,t}) \tag{2.3}$$

where $X_t = I_t(x, y)$ is the current pixel value and $\eta\left(X_t, \mu_{i,t}, \Sigma_{i,t}\right)$ is a Gaussian probability density function with the mean and covariance $\mu_{i,t}, \Sigma_{i,t}$ respectively. Here $\omega_{i,t}$

are the weights for each component that sum to one. The weights are updated according to:

$$\omega_{i,t} = (1-\alpha)\omega_{i,t-1} + \alpha(M_{i,t}) \quad (2.4)$$

where α is a user-defined learning rate in the range (0,1], i is an index to each component in the mixture, and t is the current time or frame.

A binary variable $M_{i,t}$ determines whether or not to add α to the updated weight value. If the pixel is a match to a component, then α is added, increasing $\omega_{i,t}$ during the update, otherwise the second term disappears and the weight is reduced by the update. All weights for each model are normalized to sum to 1 at the end of the update process.

The mean for each component is updated at each frame t using:

$$\mu_t = (1-\rho)\mu_{t-1} + \rho X_t \quad (2.5)$$

Similarly the variance is updated at each time t by:

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^2 \quad (2.6)$$

The speed at which our model is updated depends on ρ and the fixed learning rate parameter α from (2.4) where ρ is simply α scaled by the inverse weight of that component.

$$\rho \approx \frac{\alpha}{\omega_{i,t}} \quad (2.7)$$

To determine if a pixel value is part of the background, one common approach previously described is to set some threshold distance from the mean within which a pixel is called as background. For a background model with K Gaussian components, you simply find the component whose mean has the closest match to the pixel value in question. The resulting output will be a binary image where we set background pixels to 1 and foreground to 0. Using only a binary indicator to differentiate foreground from background is useful in the case of simple, easily separable foreground and background objects. The problem is that useful information about the closeness of the match between the pixel value and the model is discarded; information that may be needed if there is a lot of noise in the resulting output. For this reason, here the approach is modified and instead a confidence value is generated based on how far away the pixel values are from the mean. The confidence is then normalized by the standard deviation. This allows a softer way of making a determination about the pixels that make it into the foreground model and it implies that there is less importance on precise tuning the GMM parameters. The final decision about something being a hit or non-hit can be postponed until later in the process, consistent with Marr's Principle of Least Commitment [9]. The confidence output can be computing now using:

$$FG(x,y) = \frac{\min\left(abs\left(\frac{(X-\mu)}{\sigma}\right), n\sigma\right)}{n\sigma} \tag{2.8}$$

Where μ is the mean of the matching background component (note if there is no matching background component, then the confidence is 1 that the pixel belongs to the foreground. n is some number of standard deviations σ chosen to consider; one common value used here is 2.5σ). The standard deviations and means are initialized to some value depending on the type of data being used. Typically for grayscale data, the system is initialized with μ for each pixel's components set to some random value in the grayscale range [0,255] and σ is initialized to a fixed value in the same range; in previous work 30 has been found to work well [10].

The standard deviation can be seen as a measure of confidence in the accuracy of the background component. The implementation in [8] reflects this in that components that match pixel values have their standard deviations reduced. Normalizing the distance between the test pixel from the mean using the standard deviation provides a measure of how close the pixel is to the model and how much trust to place in the model. A large standard deviation suggests a low confidence in the model component itself. This will result in lower confidence value, even if the pixel value is close to the mean. Similarly, a large distance between the mean and test pixel could still produce a high confidence value providing that the standard deviation is small enough. In the confidence definition above, the maximum confidence is $n\sigma$ (standard deviations) and this is normalized by $n\sigma$ to ensure a [0,1] range.

Using this confidence measure, output data for all pixels in a given window is generated. The output of interest is the foreground which in an ideal case would contain only target objects. The image shown in Figure 2.6 below shows a single frame over which the algorithm was run for a fixed window size covering only a subset of the image frame. In this image the intensity values correspond to confidence that a given pixel is foreground. This could also be interpreted as a measure of the local change as a very high confidence indicates that this pixel value has not been seen recently and, therefore is anomalous. Collections of such pixels often correlate with the location of a buried target as is the case below.

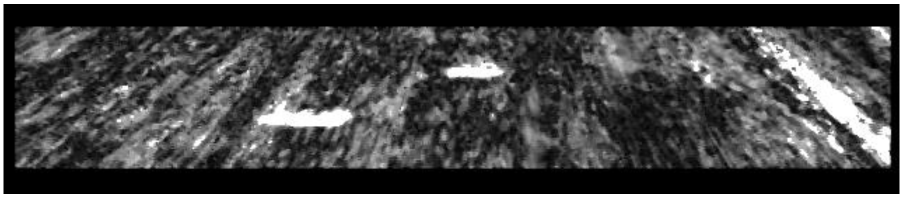


Figure 2.6. Output from the GMM shown as a scaled image. Higher intensity areas correspond to higher confidence that the pixel is foreground. The two bright disks correspond to the targets shown in Figure 2. This image represents a detection window of 640x300 pixels.

An alternative method to generate the pixel confidence value is by calculating the log likelihood of the pixel value based on its probability. The likelihood can be obtained by evaluating the Gaussian *pdf* using the current matching component mean and standard deviation.

2.3 Texture Feature Analysis

Various texture features will now be examined, but first an overview is given of a common dimensionality reduction technique used the experiments called Principle Component Analysis or PCA.

2.3.1 Principle Component Analysis

In order to speed up the texture analysis algorithm and to make visualization and debugging easier it was decided to use Principle Component Analysis (PCA) to reduce the dimensionality of the data. The original feature vector is composed of multiple features (see Section 2.3.2-2.3.4) and has a dimensionality of around 54 depending upon which features and settings are being used. Though 54 dimension vectors are computable with current systems, it tends to be slow, and for debugging and testing is hard to visualize and evaluate.

The premise behind PCA is that not all of the information in a feature vector is necessarily that useful in providing information for classification. It is possible to project the data-points down onto a vector with a direction that maximizes the variance of the projected data [11]. It turns out that the direction that maximizes the variance is the same as the eigenvector (of the covariance matrix of the data) with the largest eigenvalue. If the data-points are projected onto this line, then the line rotated so it corresponds with the x-axis, say, then only one dimension is now needed to store the feature data.

The data in the example below has been reduced from 2-dimensions to 1, but unless less all the original data lay on a line in the direction of the principle component, there is an error associated with the discarding the other dimension. When using PCA with the texture data, dimensionality can be reduced from around 54 to 2D or 3D. It is important to note that the aim of PCA is to make sure that the first principle component has the highest possible variance [12]; PCA is not guaranteed to find the projection that preserves the best separability between classes. It is important to be clear that this is not feature selection and all the feature data is used in the PCA (though there are PCA-based feature selection techniques such as Principle Feature Analysis [13]). Further, there is no reason to limit the number of principle components used to 1. In fact, the number chosen depends on the acceptable error rate and the percentage of the data variance attributed to each component.

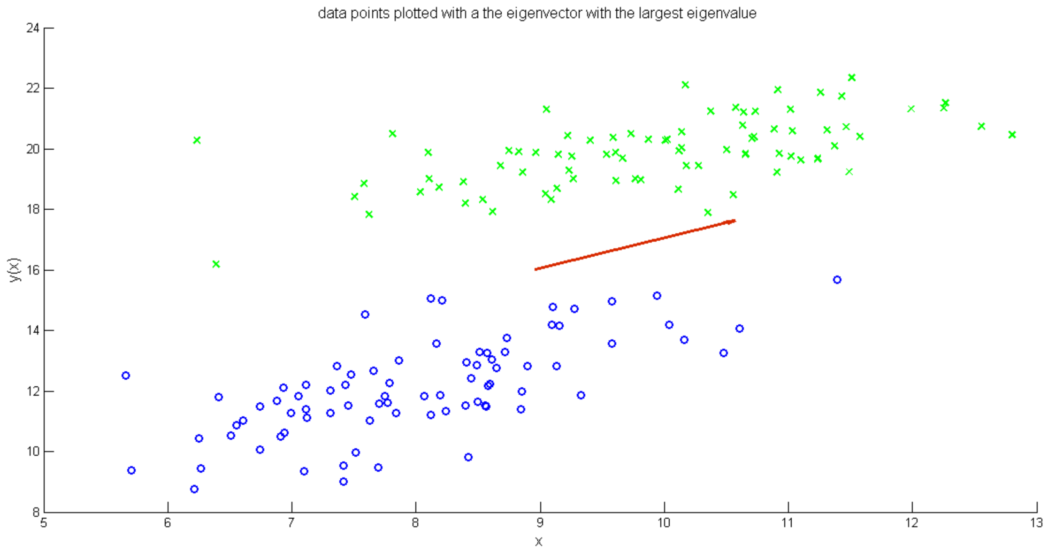


Figure 2.7. Plot of synthetic data demonstrating variance mostly along the eigenvector with the largest eigenvalue which is plotted in red at the mean of the dataset.

PCA can also be thought of as transforming the data into a form represented by orthogonal vectors where the vector points along a line defined as passing through the mean and which has the lowest least squares sum of differences.

Because of the reliance on correlations between the different dimensions in the data, the first step in PCA is to normalize the data and then to look at the covariance between each dimension (note: the method described here is known as eigenvalue decomposition, but PCA can also be achieved using singular value decomposition [11]).

The covariance matrix for the data is generated as follows using the definition of covariance between two variables, X_i and X_j:

$$Cov(X_i, X_j) = E[(X_i - E[X_i])(X_j - E[X_j])] \tag{2.9}$$

Using (2.9), a matrix can be built where each entry corresponds to a particular covariance between X_i and X_j where i and j are the row and column indices respectively. Next, the eigenvectors and eigenvalues of the covariance matrix are found. The eigenvectors with the greatest corresponding eigenvalues are chosen as the new basis vectors upon which the original data is projected. If the number of basis vectors is less than the original number of dimensions, then the projected data has dimensionality determined by the number of eigenvectors chosen for the projection. For example, 99% of the variance is accounted for by the first principle component in the example data used for Figure 2.7. After the PCA, the data can be represented in transformed form with just one variable. Finally, there is the option of recovering the original data, less the component with the least variance. In effect this acts as a method of compressing the data as shown in Figure 2.8.

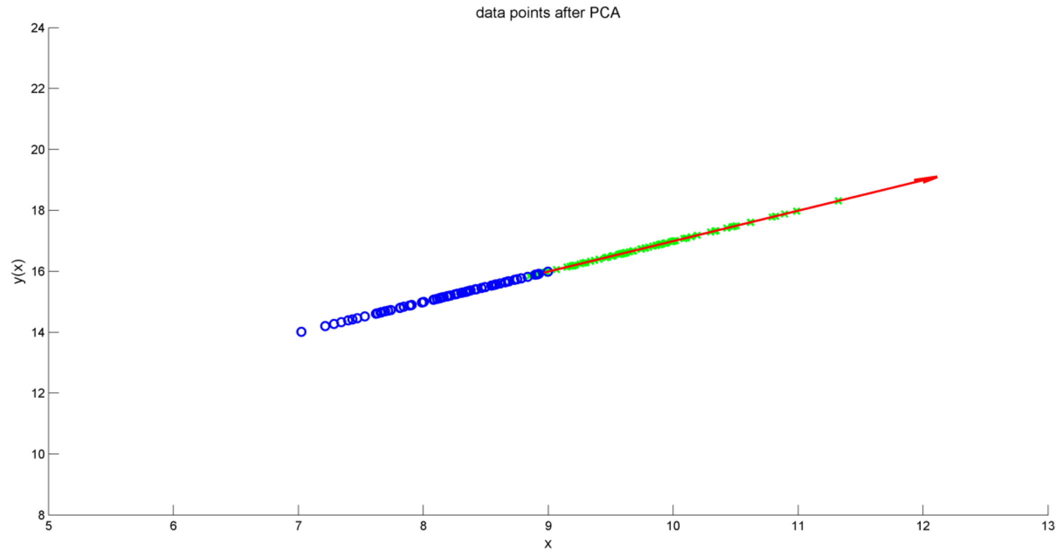


Figure 2.8. Data compressed with PCA then converted back to original coordinate system. Shown also is a vector in the direction of the original principle eigenvector.

2.3.2 Local Binary Patterns

One aspect of texture can be described by the local variation in contrast and gray-level in a region around a pixel. We want to measure the contrast between pixels without concern for the actual grey-levels involved; a useful measure would therefore be invariant to the actual grayscale levels. In other words given three texture images A, B and C, where B is simply A + p where p is some integer and C is the negative or inverse of A, we would expect all three images to still demonstrate the same texture, and so any measure we use on them should return similar results. The image B in this example could represent the same texture in another part of the image with different illumination.

The group of methods that measure local binary patterns (LBP) have this invariance to actual gray levels, and several of them are also rotation invariant which is another important property when dealing with texture. Another attractive feature of LBP is that compared to some measures it is computationally inexpensive [14].

The basic idea behind LBPs is that pixels are compared to their neighbors based on their gray-level. One simple method of doing this is to take each pixel and compare it to its 8 neighbors by thresholding the neighbors by the center pixel value. If a neighbor pixel value is higher than the center pixel we record a 1 for that position, else we record 0 if a neighbor is lower. This pattern of 1's and 0's is what gives rise to the name of this technique.

The pattern of ones and zeros when read in order form a binary number that can be converted to decimal. The actual binary pattern produced is dependent on which neighboring pixel the sequence starts at. For a sequence of n neighbor pixels (where so far we have considered n=8 neighbor pixels) we get 2^n possible patterns. For any given pattern, assuming the pixels are read sequentially, then each different starting point is equivalent to some shift of the basic sequence (see Figure 2.9). This shift is equivalent to a rotation of the bits often seen in computing and digital logic. This rotation property not only shows itself with choice of starting location for the binary number, but is also associated with rotation of the image itself. Two identical textures could have the same pattern, only rotated by some number of bits.

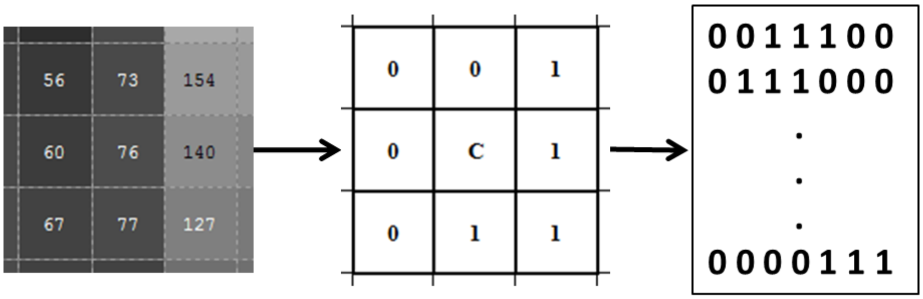


Figure 2.9. Example of a local binary pattern formed by a group of 9 pixels. C marks the center pixel used for the thresholding.

To correct this and to make the binary identifier rotation invariant, each pattern is rotated enough times to ensure that the smallest binary number is generated. Basically, the bits are rotated until the set bits (1's) are in the place of least significance.

Recall that number systems weigh their importance via position in the sequence; this is true of any number system. Subscripts are used to designate the number base or radix and superscripts for powers. For example 128_{10}: 1x100 + 2x10 + 8x1 or put another way in radix or base 10 we have $1x10^2 + 2x10^1 + 8x10^0$. Similarly in binary, or radix 2: 101 is $1x2^2 + 0x2^1 + 1x2^0$ to give us $101_2 = 5_{10}$ where the subscripts give us the radix. It is easily seen that the weight given to each digit depends on its position in the sequence. For any sequence of given digits it is possible to rotate them so the largest weights are in the least significant positions.

The problem with this approach is that although it is invariant to rotation, quite a lot of information is discarded or ignored. For this reason in [14] an alternate measure called Uniform LBP was introduced. Uniform LBP basically counts the number of 0 to 1 transitions in the sequence, if this number is less than or equal to 2 transitions (uniform-2), then an integer label is applied to that pattern representing the number of set bits (1’s) in the sequence. The motivation here is that sequences of 2 or less transitions are of particular interest as they seem to correspond to feature detectors [14]. For example, the LBP patterns centered on a dark line against a light background in would show a transition from zero on a part of the line, to one for part of the background, and then back to zero as the line is crossed again Figure 2.10. These transitions represent a region going from a bright spot to a dark spot or vice versa and can represent edges or corners, etc. It was found in [14] that these 2-transition sequences represented 90% of the patterns found in texture images.

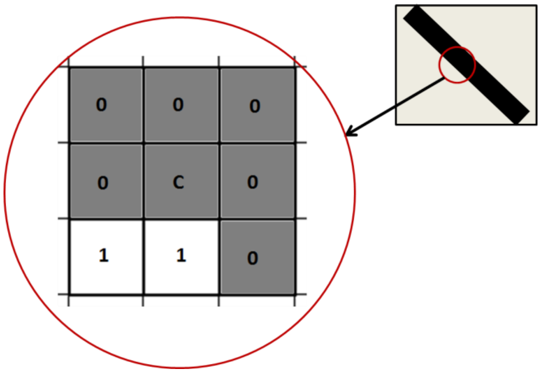


Figure 2.10. Example of an LBP pattern centered on the outside edge of a dark line on a brighter background. The LBP pattern transitions from 1 to 0's then back again in a circle.

When used on an image segment, the uniform LBP values over the entire segment can be histogrammed into P bins, where P is the number of patterns possible with a given neighborhood size. The LBP's that have more than 2 transitions are binned together in one large bin at P+1 [14]. This histogram can then be used as a feature vector where each element in the vector corresponds to the frequency from the histogram.

This algorithm can be run over an entire image or image segment either returning an LBP image where the center pixels are set to the integer value of the (smallest that can be represented by rotating the bits) binary number (Figure 2.11) or a histogram of all the values (Figure 2.12).

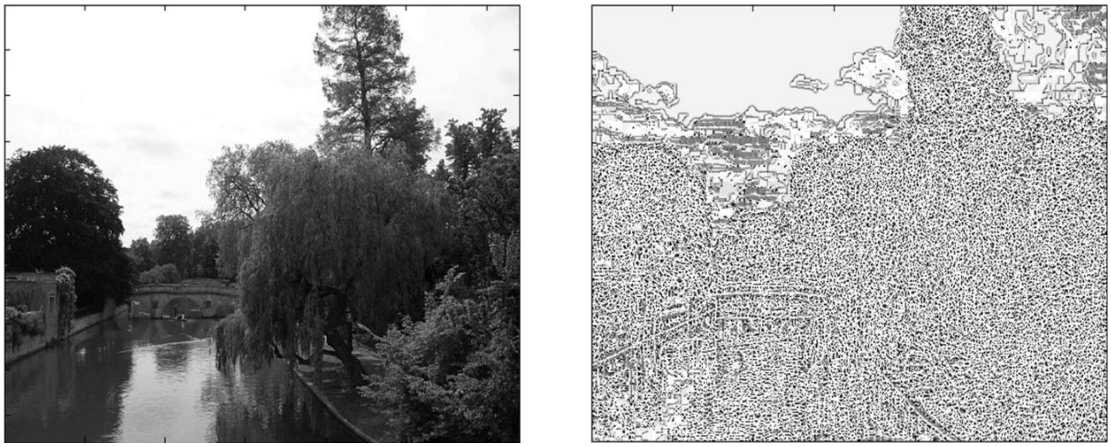


Figure 2.11. Example of LBP applied over a whole image where the pixel values in the original (left) have been replaced by the binary pattern value to form an LBP image (right).

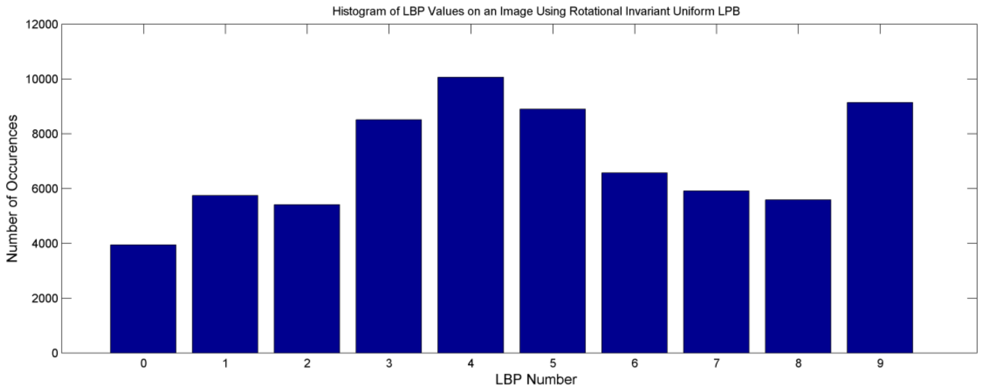


Figure 2.12. Histogram output from rotation invariant, Uniform LBP for a segment of the image of the river above. The bin at 9 is not an LBP, but combines all patterns with more than 2 transitions in a single bin. The other 9 bins represent all the possible patterns with 2 transitions in a neighborhood of 8 pixels.

2.3.3 Histogram of Oriented Gradients

An important feature of the human visual system is the ability to detect edges and corners. For example, if shown an image of bricks or grass texture humans can identify it in part due to the lines or edges in the images. If all information except the edges is removed, the textures are still recognizable (see Figure 2.13).

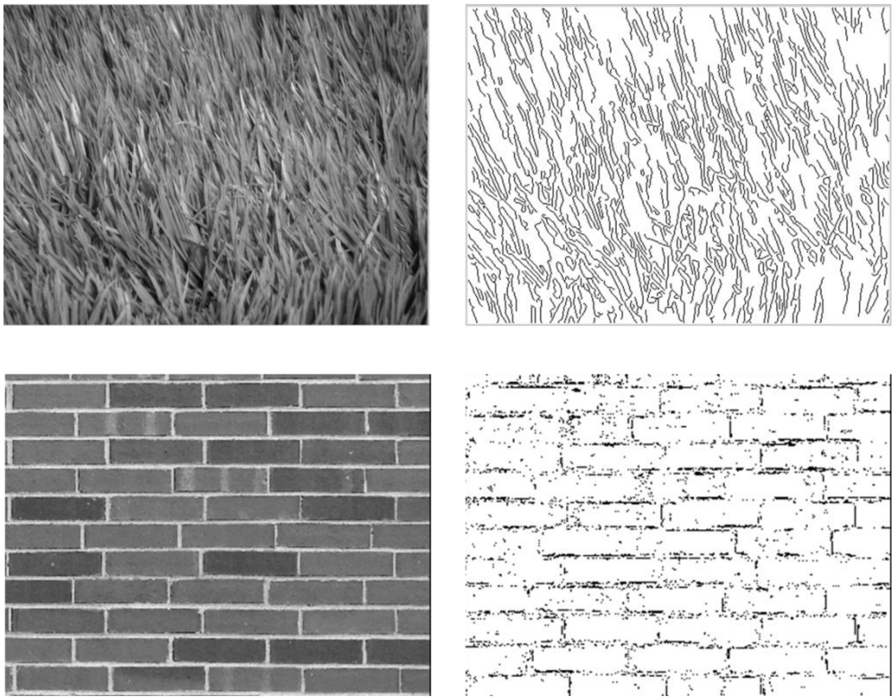


Figure 2.13. Example of the information still present after detecting edges and discarding all other data in the image. Original images on left taken from [15].

One way of detecting the edges in an image it to find the gradient vector with components

$$\nabla I(x,y) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) \tag{2.10}$$

In the case of an image which is discrete, an approximation for the partial derivatives at each pixel location (x,y) is as follows:

$$\frac{\partial I}{\partial x} = I(x-1) - I(x+1) \tag{2.11}$$

$$\frac{\partial I}{\partial y} = I(y-1) - I(y+1) \tag{2.12}$$

This is the discrete, 1-D centered, point derivative for x and y and is achieved by filtering with a simple mask: [-1 0 1] for x and $[-1\ 0\ 1]^T$ for y [16].

These functions are applied separately as a filter to create two new images for the x and y partial derivatives. Once this is done, the two images are combined to create a gradient magnitude image and a gradient direction image. For gradient magnitude the following is used:

$$|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{2.13}$$

And for the gradient direction:

$$\theta = atan\left(\frac{\frac{\partial I}{\partial y}}{\frac{\partial I}{\partial x}}\right) \tag{2.14}$$

Figure 2.14 below demonstrates the partial derivative images as well as the gradient magnitude images for the bricks texture shown above in Figure 2.13:

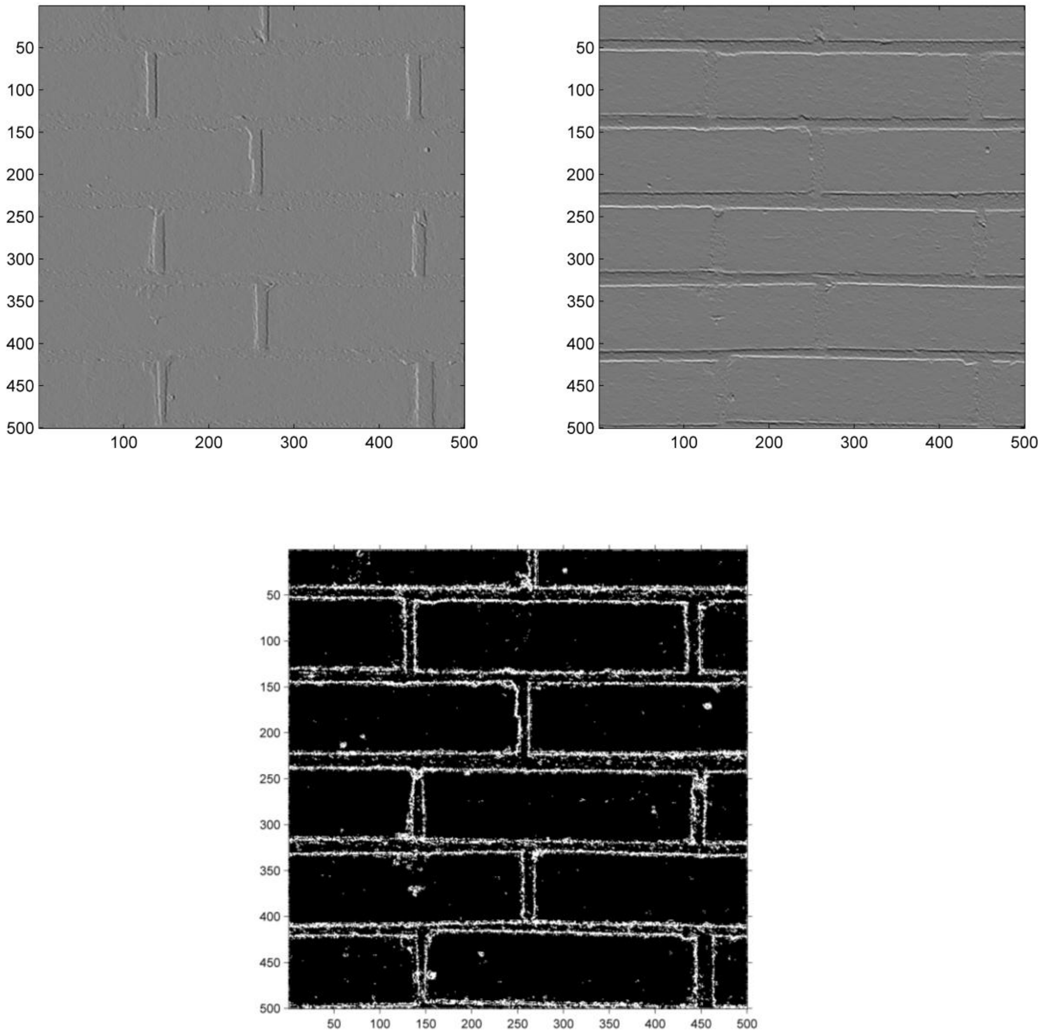


Figure 2.14. Example of the bricks texture showing gradient images in x-direction (top-left) and y-direction (top-right) and the gradient magnitude image (bottom).

Once the gradient image has been calculated, it is possible to generate statistics based on the magnitude and orientation of the gradients at different points on the image. These statistics will hopefully be unique to particular textures. One such statistic, known as Histogram of Oriented Gradients (HOG) [17] involves dividing the image into a number of cells (Figure 2.15) and building a histogram of the gradient directions in each cell. The magnitude information is also important and is used in HOG as a weighting factor on each direction in the cell. Therefore the final value for each histogram bin b is the sum of the gradient magnitudes w for the directions d that map to that bin. If n pixels have gradient orientations that fall into the range for bin b:

$$b = \sum_{i=1}^{n} w_i \tag{2.15}$$

Once the cell histograms have been calculated they are locally normalized by aggregating groups of cells together into blocks. These blocks are overlapped, so a cell will appear in more than one block [17] . All the histograms in the block are normalized by taking the vector v containing the bin values for all the block's histograms and taking the L1-norm to yield v_N [17]:

$$v_N = \frac{v}{\|v\|_1 + \varepsilon} \tag{2.16}$$

ε is a regularization constant set to a small value such as 0.2 as per [17].

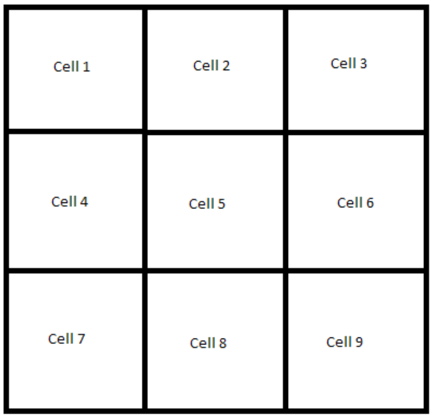


Figure 2.15. Cell structure of a single HOG block. Blocks overlap, so the cells here can be members of more than one block. Histograms for each cell are normalized using all the cells in the block so there will be multiple normalized values for each cell (one for each block it is a member of).

The final feature vector contains the histograms from the cells. Each cell has multiple histograms associated with it due to the different normalizations. For example, if there are 2x2 blocks used with the cell structure in Figure 2.15 and these blocks overlap by 1 cell, then cell 5 will generate four different values, one for each block normalization. Therefore there will be four histograms for that cell in the final feature vector. Figure 2.16 below shows one set of histograms for a group of cells for easier visualization

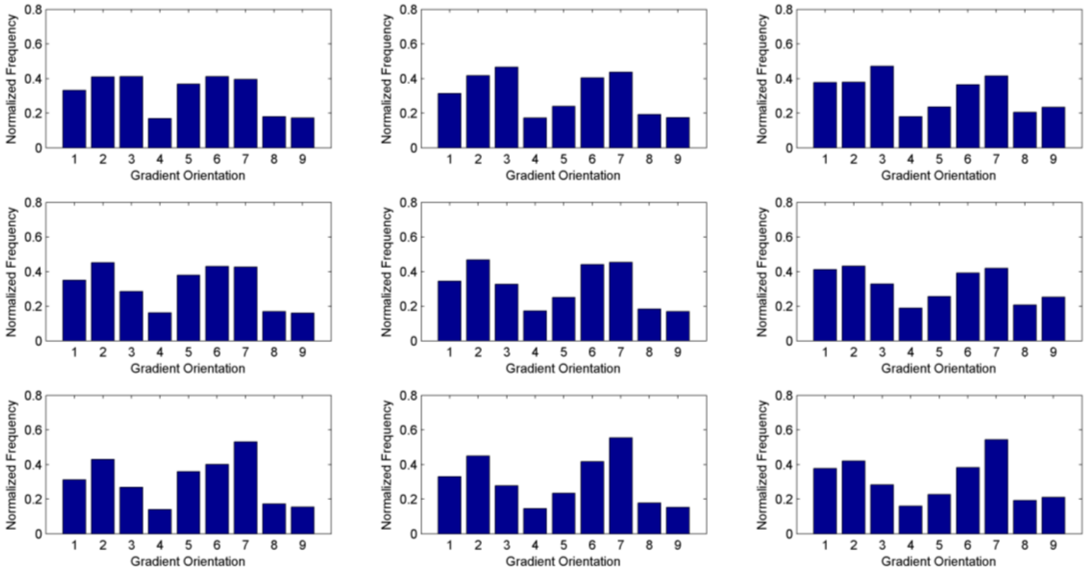


Figure 2.16. Shows one set of normalized histograms from a 3x3 grid of cells.

2.3.4 Gray-level Co-Occurrence Features

One of the earliest means of measuring texture was to examine the relationship between neighboring pixels in an image by recording the distribution of gray-levels in a matrix. The Gray-Level Co-Occurrence Matrix (GLCM) is a matrix that records the occurrence of particular relationships between adjacent pixels [18]. For example, the matrix may record how many pixels of value *a* have a horizontal neighbor *b*. So, for a matrix A, the entry for A_{ij} would record the number of times a pixel with value *i* has a neighbor value *j* in the horizontal direction. In fact, the GLCM is required to be symmetrical and so the relationship is recorded twice: once as A_{ij} and once in A_{ji}, this can be achieved by adding the matrix to its transpose [19]. The final matrix size would then be N x N where N is the number of gray-levels in the image though often images are first scaled to reduce the number of gray-levels [19].

Once the GLCM matrix has been filled, the values are normalized by dividing by the total count of all the relationships which is simply the sum of all values in the matrix [19]. Now instead of an integer count at A_{ij}, there is a probability $p(i,j)$. This normalized GLCM can then be used to calculate various texture features including contrast, correlation, energy and homogeneity.

There are several measures for contrast and most examine the diagonal elements of the GLCM. As the diagonal elements represent pixels where the neighbor has the same value (such as (1,1), (2,2), (3,3), etc.) then large sums in the diagonal elements represent a lot of self-similar neighbor relationships and so a lack of variation in the image [19]. Another measure, correlation, is very similar to autocorrelation and looks for a relationship between a pixel and its neighboring value. Energy is just the sum of the squares of all the elements in the matrix and homogeneity measures how close the distribution of pixel values in the matrix is compared to its diagonal [20].

The GLCM measures used in this thesis are built into the MATLAB Image Processing Toolbox and are calculated using the following equations. Again, $p(i,j)$ is the value found at index (i,j).

Contrast:

$$\sum_{i,j} |i-j|^2 p(i,j) \tag{2.17}$$

Correlation:

$$\sum_{i,j} \frac{(i-\mu_i)(j-\mu_j)p(i,j)}{\sigma_i \sigma_j} \tag{2.18}$$

Energy:

$$\sum_{i,j} p(i,j)^2 \tag{2.19}$$

Homogeneity:

$$\sum_{i,j} \frac{p(i,j)}{1+|i-j|} \tag{2.20}$$

The GLCM features are simple and quick to calculate and several of the GLCM features such as correlation are independent of the others [19]. Another useful aspect is that there are a many different methods of calculating the GLCM itself; for example you can vary the neighbor direction, the distance to a neighbor and the number of gray-levels used. Many calculations or statistical measures used on the GLCM qualify as texture features and may be useful in classification. Finally, the GLCM measures are second order measures that operate on the GLCM and so should hopefully reveal relationships not apparent using measures calculated directly from the image.

2.3.5 Fractal Dimension

Fractals are geometric shapes that show properties of self-similarity at all scales. No matter how much you zoom in on a fractal you will see the same patterns repeated. The fractals can be in the form of a 2D fractal curve or in the form of a 3D fractal surface. It is also possible for a 3D fractal surface to have a perimeter which is a fractal curve. The interesting thing about the fractal curve that it is infinite in length [21]. The measures used here concentrate on the fractal dimension of the fractal surface of the image regions. First 2D fractal measures used on binary images will be discussed, next these techniques are extended to the three dimensional surfaces represented by the IR images.

It is sometimes difficult to define what is meant when describing the word texture as applied to images. Two major aspects do come to mind however, one is self-similarity and the other is surface roughness. Self-similarity can be seen in repeated structure elements (textons) [16] over an area and surface roughness can be said to be a measure of the size or density of the textons in a given region. As self-similarity is also a feature of fractals, then it may be advantageous to use fractal measures on the image segments. One such measure that corresponds to human perception of surface roughness is fractal-dimension [22].

Surface roughness is one measure; another way of thinking of fractal dimension is as a measure of how well an object fills the space or dimension within which it is contained. For example a solid square will fill the 2D space within its boundaries completely. Conversely, an image of a sponge is full of holes and therefore does not completely fill the space. Following this intuitive idea the space filling property of a binary image can be measured by placing a grid over the image and counting the instances where a grid square contains a part of the image (pixel values of 1) and ignoring those where no part of the image is contained such as a square containing pixels of value zero. This idea can be generalized in 2D by saying that only *significant* pixels are considered when counting squares. If the size of the grid squares is varied in increments, and if at each step the number of squares with significant pixels is counted, then using the following equations an estimation of fractal dimension can be made. This measure is known as the box-counting dimension and the results for simple 2D fractals are comparable to the known calculated dimension [21]. In order to use 2D box-counting on images they have to be converted to binary images containing only ones and zeros. This flattens the 3D surface down onto the plane using some threshold.

Fractal dimension of an object can be measured by calculating how many scaled copies of an object or surface are needed to cover the original. For example, if we scale an object in each of its dimensions by l, it then takes $N(l) = l^D$ of these smaller objects to cover the original (unscaled) object.

To recover the dimension, D of the original object we notice as before that:

$$N(l) \approx l^D \tag{2.21}$$

So by taking the log of each side (changing from base l using $\log_l N(l) = \frac{\log N(l)}{\log l}$)

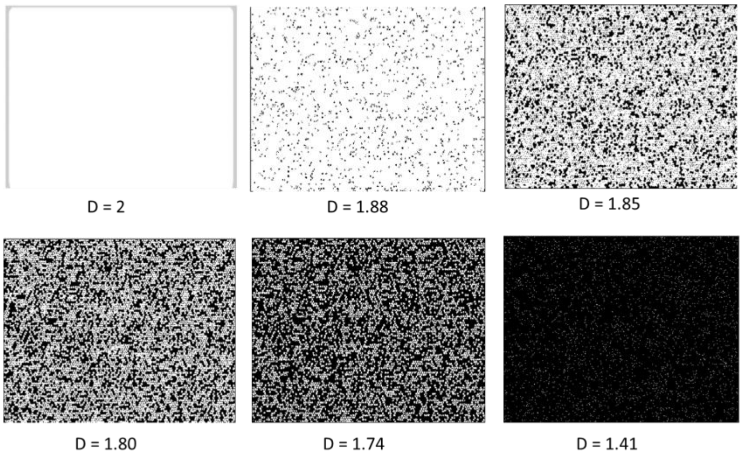
we get for the dimension D:

$$D = \frac{\log(N(l))}{\log(l)} \tag{2.22}$$

To get the fractal dimension we take the limit as $\varepsilon \to 0$ where ε is the size of the scaled objects needed to cover the original.

$$D = \lim_{\varepsilon \to 0} \frac{\log N(\varepsilon)}{\log \frac{1}{\varepsilon}} \tag{2.23}$$

In the case of box-counting, $N(\varepsilon)$ is the number boxes are needed of size ε to cover the image or object [21].



D = 2
D = 1.88
D = 1.85
D = 1.80
D = 1.74
D = 1.41

Figure 2.17. Example images showing how fractal dimension D decreases as the space filled decreases. Here the significant pixels are white.

It should be noted that often the dimension calculated in this way is only proportional to the true dimension. In that case $N(l) = kl^D$ where k is some constant of proportionality and the dimension may be more accurately represented as $D = \frac{\log\left(N(l)\right) - \log(k)}{\log(l)}$.

There are several other measures of fractal dimension some of these are described in [22] with the equations given as follows:

The information dimension is:

$$D_I(F) \equiv \lim_{\varepsilon \to 0} \frac{I(\varepsilon)}{-\log_{10} \varepsilon} \tag{2.24}$$

$$I(\varepsilon) \equiv \sum_{i=1}^{N(\varepsilon)} -P(\,\varepsilon, i) \log_{10}(P(\varepsilon, i)) \tag{2.25}$$

For the information dimension the knowledge gained by knowing which box in the grid a particular point from the set F appears in is used and is measured using $-\log_{10}(P(\varepsilon, i))$. Here $P(\varepsilon, i)$ is the *natural measure* or probability of a point being in (populating) box *i* with box size ε. In the case of pixels this measure takes into account their spatial distribution which could be useful, especially for texture.

The correlation dimension is given by as follows where $s(i, j) = \|i - j\|$ which is the Euclidean distance between a pair of points i and j.

$$D_C(F) \equiv \lim_{r \to 0} \frac{\log_{10}(C(r))}{-\log_{10}(r)} \tag{2.26}$$

$$C(r) \equiv \frac{number\ of\ pairs\ of\ points\ (i, j)\ with\ s(i, j) < r}{(total\ number\ of\ pairs\ of\ points\ (i, j))} \tag{2.27}$$

The idea here is to measure the Euclidean distance between all the points in the set and count the percentage less than some distance r. The result can be plotted for multiple values of r and the fractal dimension calculated from the slope as shown in Figure 2.19.

Using the box-counting method the fractal dimension for at different scales (values for ε) can be plotted. This can be done in two ways. First, the number of boxes to cover the object versus the size can be plotted on a log-log plot. If the image under analysis follows the power law in equation 2.18, then this plot should show a straight line with a slope equal to the fractal dimension [23]. Second, it is also then possible to plot the slope of the line directly by taking the derivative of the ratio shown in (2.19) and plotting with respect to the box size. It should be noted that the plot is not always as well behaved as shown in Figure 2.19, and for some images it can be hard to fit a line. Often it is the points at either extreme of the scale that are the outliers and these can be trimmed before fitting the line. An example of calculating the fractal dimension using box-counting is shown below in Figure 2.19 using the Sierpinski triangle (Figure 2.18) which where the dimension can be calculated as :

$$D = \frac{log3}{log2} = 1.585 \tag{2.28}$$

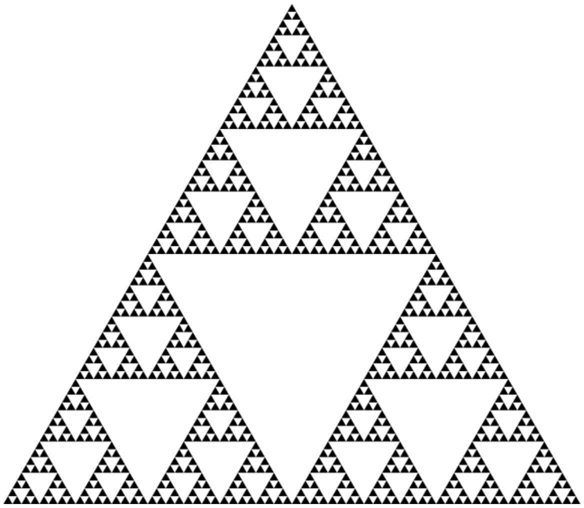


Figure 2.18. Example of a well-known fractal called the Sierpinski Triangle. Note that you can clearly see that it takes 3 triangles scaled by ½ to cover the original.

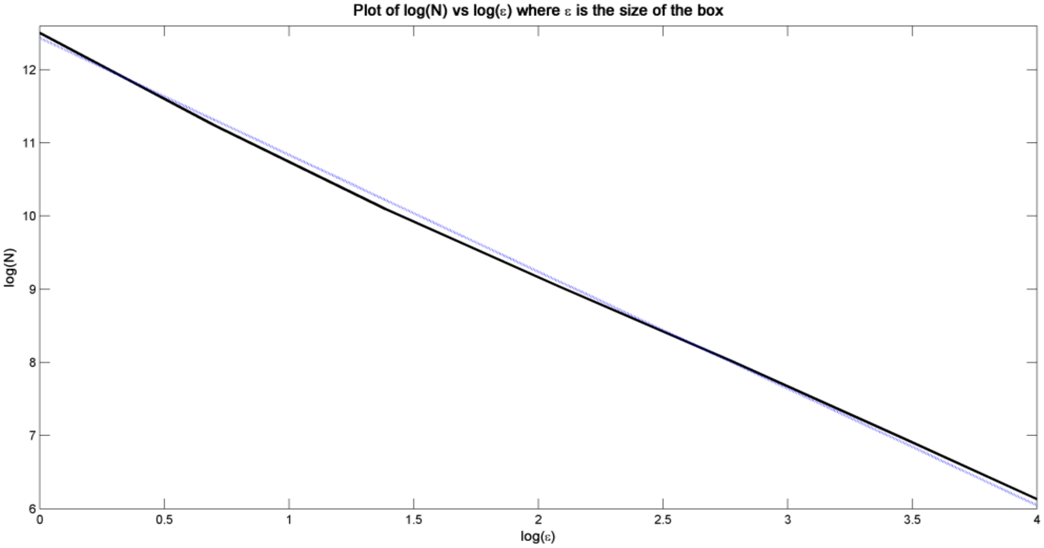


Figure 2.19. Output of the simple 2D box-counting example using the Sierpinski triangle as a test input. The dark line is the original plot and the lighter line is the line of best fit. The magnitude of the slope of the best fit line corresponds well with the known fractal dimension of 1.56.

In Figure 2.19 above it can be seen that (absolute) value of the best-fit slope of 1.597 corresponds reasonably well to the directly calculated dimension [24].

As the infrared imagery under study is a 3D dimensional representation with x, y and grayscale values for each pixel basic 2D box counting will not work. There are several solutions to this problem. One solution is to reduce the grayscale image into a binary image. This can be done by thresholding, edge detection or dithering or some other means. All these methods require the setting of some threshold or other parameter and the results are often unsatisfactory. These methods work well for simple objects whose structure tends to be in the 2D plane rather than as a surface in 3D. Below in Figure 2.20 is an example of a color image of a leaf thresholded into a binary image. As you can see, the overall structure is still clear.

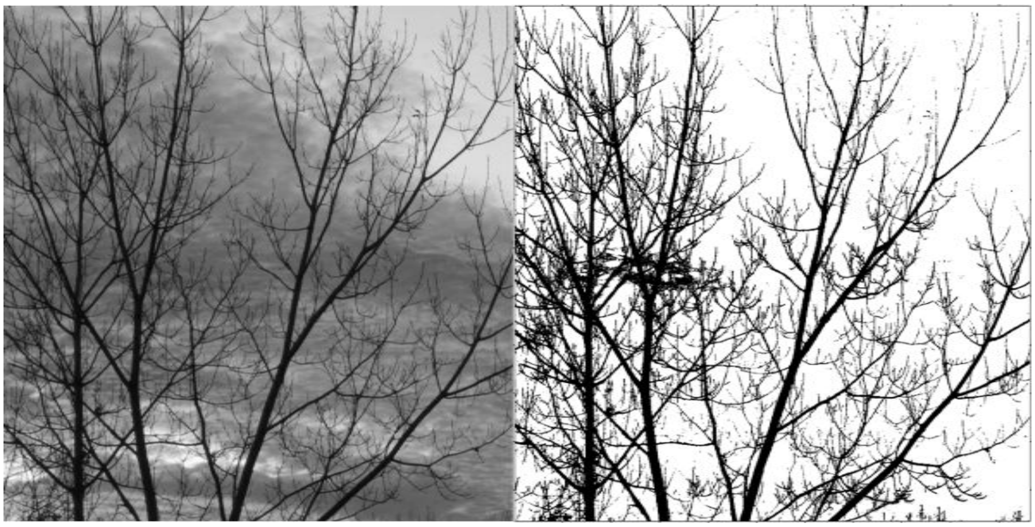


Figure 2.20.Thresholded image where most the information is contained in the x-y coordinates and the main structure remains after thresholding. This type of image is well suited for 2D box-counting.

In Figure 2.21-2.23 show examples of converting grayscale to binary imagery by thresholding, dithering and edge-detection respectively. It can be seen that much of the detail is lost compared to the 3D surface plots in Figure 2.24.

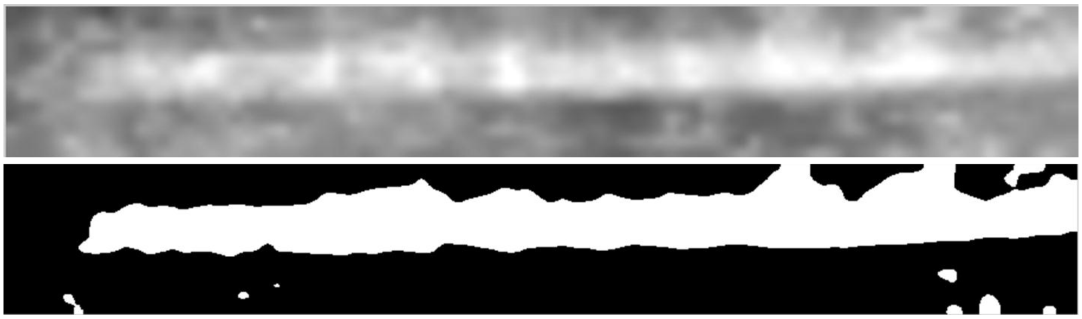


Figure 2.21. Thresholding of a target image from the IR. None of the detail in the center of the target (top) is retained in the binary image (bottom).

As can be seen from the thresholded image above, not much of the structural detail remains. This is true of any of the many different threshold levels attempted. As thresholding is unsatisfactory, dithering was tried instead. Dithering is a method of representing grayscale images in binary format using different densities of pixels. Examples of dithering can often be seen in low quality images used in newspapers.

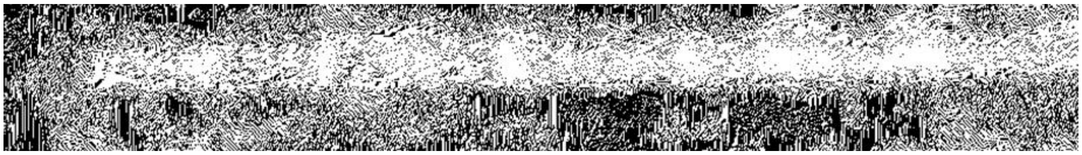


Figure 2.22. The same target shown above, this time the conversion to a binary image was done using dithering.

Again, the results did not preserve the detail in the original grayscale image, at least not in a consistent way. Dithering also introduces its own problems with parameters that have to be set.

One last attempt at finding a good binary representation of the grayscale target images for use with 2D box-counting was to use edge detection. Here the results in Figure 2.23 again show a loss of detail.

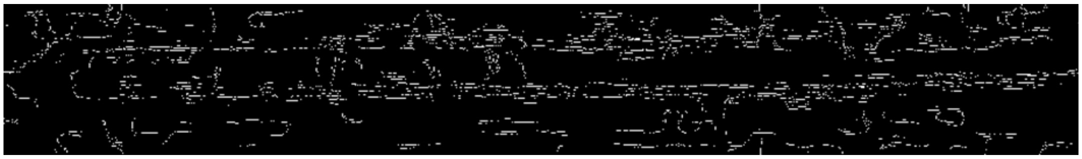


Figure 2.23. The target image from above binarized using Canny edge-detection.

If the edge detection was refined, then perhaps it could be used with some means of measuring the fractal dimension of the perimeter or silhouette such as fractal Brownian motion [25]. However, as discussed below, it was decided instead to keep the images as gray-scale and use all the information present.

To demonstrate more clearly the texture in the grayscale imagery, below are two examples of known targets in the infrared imagery shown as 3D surfaces where height corresponds to gray-level.

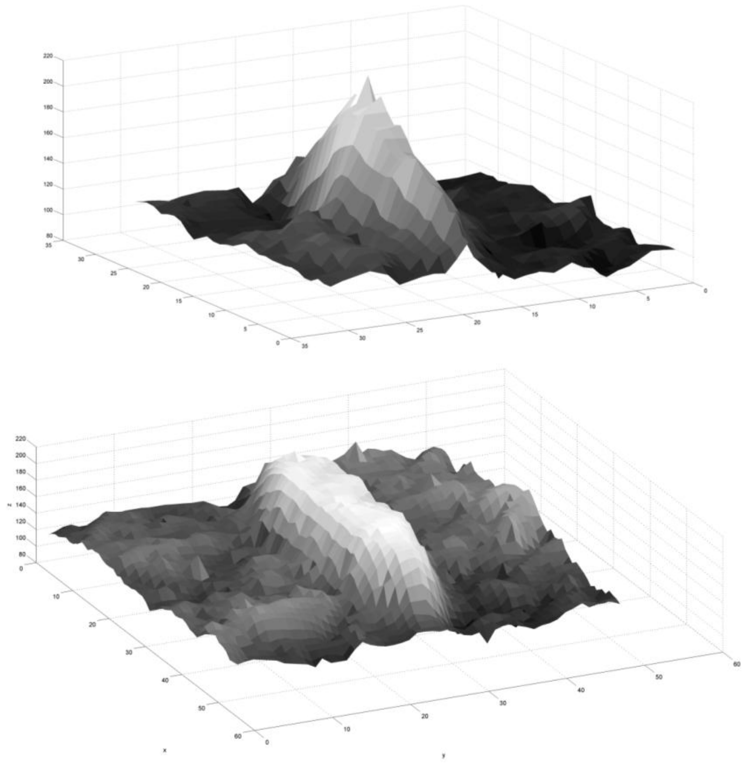


Figure 2.24. Two target patches taken from the IR imagery plotted in 3D with the vertical axis corresponding to the gray-level.

A better approach with grayscale images is to extend the box counting into 3 dimensions, in other words by counting using cubes and not images. One popular approach to this is known as Differential Box Counting (DBC) [26] which uses the distance between maximum and minimum gray-levels for different height boxes in the z-direction. The approach is described in [26] as follows:

First, divide the (x,y) plane of the grayscale image into a grid of squares as with regular box-counting. Next, build a column of rectangular boxes with length, width and height $S \times S \times S'$ respectively. The box number containing the minimum gray-level is denoted k, the box containing the maximum is called l, where the box number denotes its height or position in the column with box 1 being the base. The number *N* as shown in the numerator of equation 2.20 is now calculated as:

$$N_r = \sum_{i,j} n_r(i,j) \tag{2.29}$$

$$n_r(i,j) = l - k + 1 \tag{2.30}$$

Now (2.23) can be calculated as before for different box sizes r (ε=r in this case). Incidentally, the name of the DBC method was chosen due to the differencing carried out in (2.30).

One final fractal dimension estimator is to a technique called fractional Brownian motion (fBm) [25] which can be implemented as follows for a 1D, single slice through an image:

Algorithm 2.1: Fractional Brownian Motion

```
FOR each distance $l$ where $1 \leq l \leq N - 1$  // N is the number of pixels in the slice
   FOR x=1 to N
      Find $d_x = f(x + l) - f(x)$
   END
   $\sigma(l) = \sqrt{\left(\frac{1}{N-1}\sum_{i=1}^{N}\left(d_i - \bar{d}\right)^2\right)}$   // find standard deviation for distance data for each $l$
END FOR
$\sigma(l) = \sigma(t_s : t_e)$                    //trim so $t_s, t_e$ are the points between
                                                   //which we keep the data
Fit a line to log($l_t$) vs. log($\sigma(l_t)$      // $l_t$ are the values of $l$ used after trimming.
$D = CH$                      //find D which is proportional to the fractal dimension.
                                                   //Where H is the slope of the fitted line.
```

A similar measure can be constructed in 2D where the distance l is a vector with x and y components. In the 2D case the magnitude of the vectors is used in the log-log plots. An alternative to the 2D case is to take multiple slices from the image and concatenate those into a feature vector.

2.3.6 Scale

One important consideration when studying texture is the concept of scale. For any given texture region the texture details can vary depending on the scale at which they are examined (except fractal dimension which should be scale invariant). As an example, consider Figure 2.13 where the large scale texture is the gaps between the bricks themselves and the grid pattern they form. However, as can be seen below in Figure 2.25 when zoomed in the texture is very different.

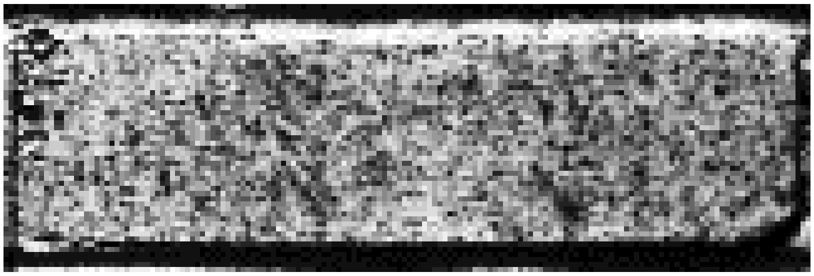


Figure 2.25. Zoomed in image of the bricks image showing another level of texture different from the macro texture of the wall.

There are two main approaches to this issue. The first approach is to examine the texture image at multiple scales by forming a pyramid of images at different resolutions. One example is the use of a Laplace pyramid. The idea behind this is to subsample the image at varying scales. We can then apply filters to the subsampled images. Below is an example of a brick wall to which a bank of oriented Gabor filters [16] has been applied to the pyramid (Figure 2.26).

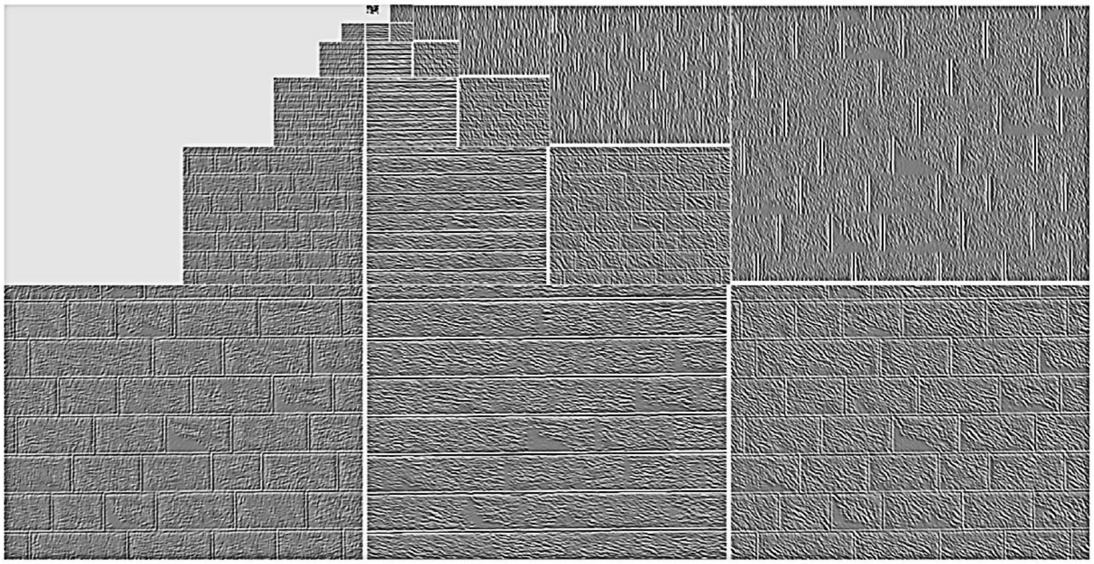


Figure 2.26. Laplace pyramid after filtering with a bank of oriented Gabor filters. The bottom left image shows the original image. The various structures revealed by each oriented filter can be seen.

The Gabor filters are oriented filters created by combining a Gaussian with a sinusoid, the orientation of the sinusoid determines the orientation of the filter and therefore the orientation of the textural features they detect [16]. Usually, a bank of filters oriented in several directions with different spatial frequencies is applied to the image. In Figure 2.26 the effect of the orientation can be seen where either horizontal or vertical lines are filtered out by filters oriented perpendicular to them.

The filtered Laplace pyramid therefore finds the orientation of the texture components at different scales. The output images can then be further processed to generate features.

After trying various scales and orientations, it was decided not to use the Laplace filter in the final version of the detection system; instead, texture samples were extracted and examined using a fixed window size.

In order to choose the best window size, some method of determining at which scale the texture stops changing is needed. In other words, find the smallest bounding box that captures the entire texture. For example, if a small window is placed over an image and the size of that window is increased it will contain more and more of the image. At some point the texture contained in the box will stop changing (i.e. be repeated). A technique for measuring this change is known as the polarity method [16]. Once the minimum sample needed to represent the texture is found, then there is no need to include more of the image than this. One analogy might be some kind of periodic wave where once a complete period is captured, a waveform of any length can be synthesized with this one repeatable unit.

The algorithm for finding polarity is given below, but the basic idea is to measure the differences in the gradients for different size windows. When the differences stop changing with the window size, then it is known that the window is large enough to capture a representative sample of the texture inside. Further increases in window size will simply capture more of the same texture. For images with multiple textures this algorithm can in theory also find the extent of a region of particular texture. For example, in the case of the bricks image the change in differences may stop when the window captures the texture of an individual brick. If the window is allowed to increase beyond this point, eventually the polarity measure will again start changing as the window grows to be larger than the brick. Figure 2.27 below shows an example where the polarity decreases, then eventually levels off as the texture ceases to change with scale.

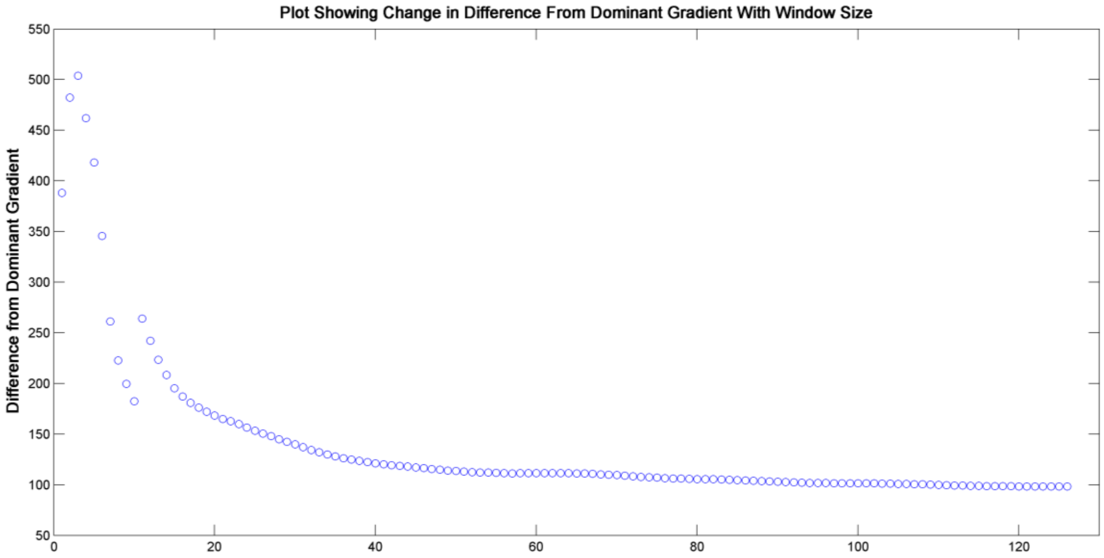


Figure 2.27. Plot showing polarity measure vs. window size on a test image. The location where polarity ceases to change significantly corresponds to the window size that captures the essence of the texture.

Algorithm 2.2: Polarity Scale

---

1) Find the average gradient direction

2) For each individual gradient take the dot product between it and the dominant gradient

3) Take the smoothed average of the positive dot products and smoothed average of the negative dot products

4) Take the difference. This will give the extent to which the gradients follow or deviate from the dominant direction.

5) When difference stops changing with increasing window size then we have the minimum scale needed to capture this texture.

6) Optional: Allow the window to keep growing until the polarity changes significantly. This is the upper bound on the texture region size.

2.4 Classification

Once the texture measures have been investigated and chosen, it is important to choose some good way of using these features to classify a particular region of the image as either a target or background. Two methods were used in this thesis: Support Vector Machines and K-Nearest Neighbor classification.

2.4.1 Support Vector Machines

The Support Vector Machine (SVM) is a popular classification algorithm based on the idea of separating two classes by a dividing boundary that maximizes the margin between the classes. In other words the line separates the two classes with the nearest points of each class being equal distance away from the boundary. A simple implementation of this idea is the optimal margin classifier [27]. The following discussion assumes that the training samples for positive class are labeled as $y = 1$ and the negative class labeled as $y = -1$.

When implementing the SVM, a determination needs to be made as to where to place the dividing hyper-plane. There are several criteria which help determine the placement as it is not sufficient to just separate the classes with some arbitrary boundary. First it would be desirable for the distance of each data point from the boundary to give some measure of confidence in the classification. For example, more confidence might be given in a classification for a point 10 units (say Euclidean) distance from the plane than is given to one only 2 units away. For this to work, the plane has to be equidistant between the nearest points from each class. These points are known as support vectors.

One major motivating factor behind the SVM is that not all the training points are needed in order to determine the placement of the dividing hyperplane. In Figure 2.28 below you can see that really only three points are necessary for determining the optimal placement of the boundary, which in this 2D case is a line. Adding more data points behind these would not change the position of the margin. This is an important feature of SVMs in that it allows unneeded data-points to be discarded creating what is known as a sparse representation of the data. Sparsity, among other things helps reduce memory overhead (compared to k-nearest neighbor for example).

More formally the boundary can be defined in terms of:

$$\boldsymbol{w}^T\boldsymbol{x} + b = 0 \qquad (2.28)$$

$\boldsymbol{w}$ is the weight vector

$\boldsymbol{x}$ are the training points

b is a bias term that can be seen as adjusting the intercept.

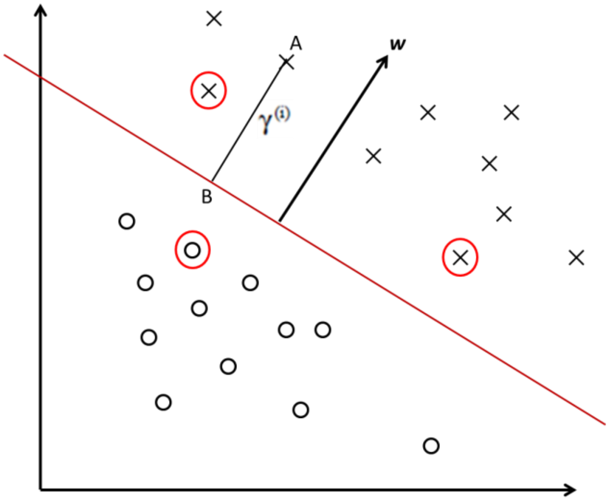


Figure 2.28. Diagram showing reasoning behind SVM. Note that this dividing line could be placed in many positions and still separate the two classes. The support vectors are circled in red.

Using this representation it can be shown that vector $\boldsymbol{w}$ is normal to the boundary and a convenient distance measure would be the distance of a point from the separating boundary in the direction of $\boldsymbol{w}$. Further, this measure can be used as the basis of the confidence level. Any point *x*, a certain distance or confidence level away can be discarded when it comes to calculating the position of the line. This would also set the corresponding weight components in $\boldsymbol{w}$ to zero.

The output of the classifier could then be a function that takes the position of the boundary in terms of $\boldsymbol{w}$ and b parameters and outputs either a positive value or negative value depending upon the classification.

A measure of the distance from the separating hyperplane to a test point $\boldsymbol{x}^{(i)}$ can be given by:

$$\gamma^{(i)} = y^{(i)}\left(\left(\frac{\boldsymbol{w}}{||\boldsymbol{w}||}\right)^T \boldsymbol{x}^{(i)} + \frac{b}{||\boldsymbol{w}||}\right) \tag{2.29}$$

Where $\gamma^{(i)}$ is known as the *geometric margin* [28] and is a measure of distance of each point from the line.

Here, $y^{(i)}$ is the target for that point (its true classification). To train the classifier then, the weights $\boldsymbol{w}$ and intersection term *b* must be found that maximize this distance whilst ensuring correct classification. Therefore if $y^{(i)} = 1$ the term $\left(\left(\frac{\boldsymbol{w}}{||\boldsymbol{w}||}\right)^T \boldsymbol{x}^{(i)} + \frac{b}{||\boldsymbol{w}||}\right) > 0$ and for $y^{(i)} = -1$ the second term must also be negative.

This function can then be solved subject to some constraints (such as the scaled minimum margins should equal to 1) by first constructing the Lagrangian, taking the dual and then solving for the Lagrange multipliers α used to calculate w using a form of coordinate ascent optimization [11].

$$\boldsymbol{w} = \sum_{i=1}^{m} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \tag{2.30}$$

As only a few training points will meet the constraints then the final result for **w** will be sparse in that it contains many zeros. Therefore the system need only be summing over a few values of $\boldsymbol{x}$, these are known as the *support vectors* and are shown in circled in Figure 2.28.

To use the weights learned to make predictions on new input values for $\boldsymbol{x}$ again use:

$$\boldsymbol{w}^T \boldsymbol{x} + b \tag{2.31}$$

here $\boldsymbol{x}$ is the new input value. Plugging in the SVM formula for *w* here you get:

$$\boldsymbol{w}^T \boldsymbol{x} + b = \sum_{i=1}^{m} \alpha_i y^{(i)} < \boldsymbol{x}^{(i)}, \boldsymbol{x} > \tag{2.32}$$

where $< \boldsymbol{x}^{(i)}, \boldsymbol{x} >$ is the inner product of $\boldsymbol{x}^{(i)}$ and **x** and this can be replaced by any function of $\boldsymbol{x}$ for example *K*($\boldsymbol{x}^{(i)}, \boldsymbol{x}$) and is known as the kernel [29].

For classification of textures both the radial basis function (RBF) and a polynomial kernel are used. The RBF is defined as follows:

$$K\left(\boldsymbol{x}^{(i)}, \boldsymbol{x}\right) = \exp\left(- \frac{\left\| \boldsymbol{x}^{(i)} - \boldsymbol{x} \right\|^2}{2\sigma^2} \right) \tag{2.31}$$

where σ is a parameter that defines the radius of the RBF [11].

The use of a kernel allows the input data to be efficiently taken into a higher dimensional space where the data may be linearly separable. The linear decision boundary in the high dimensional mapping is non-linear when brought back down into the low dimensional space of the input data [30].

The example shown in Figure 2.28 shows clearly separable data, yet it may not the case that the data is linearly separable in practice. The kernel trick allows the data to be projected into a higher dimensional space where a separating hyperplane may be found. Despite using the kernel trick, the data may still not be separable and so a modification of the linear SVM uses technique called soft margin classification which allows the SVM to tolerate misclassified training points [31]. Mislabeled training points, outliers or overlapping classes can affect the position of the hyperplane and therefore the maximum margin. If the boundary is relaxed so some points from the positive class are allowed to be on the negative class side of the hyperplane and vice versa, then in return the margin can be maximized with respect to the rest of the training data (see Figure 2.29). Despite allowing points of the wrong class either side of the hyperplane, it is still desirable to limit both the number and distance from the decision boundary of these points. One way to do this is to introduce some penalty E that takes the number and degree of transgression into account. In the simple linear SVM, if the distance of a point from the hyperplane (the margin) is a positive number, then a point $x^{(i)}$ on the wrong side can be considered to have a negative margin ξ_i. The total error for all the points is now:

$$E = C\sum_{i=1}^{N}\xi_i \tag{2.32}$$

Where C is a constant that determines how much weight to put on the error values. This formulation for the error makes intuitive sense as it accounts for the number of points with negative margin, and how far they are from the hyperplane.

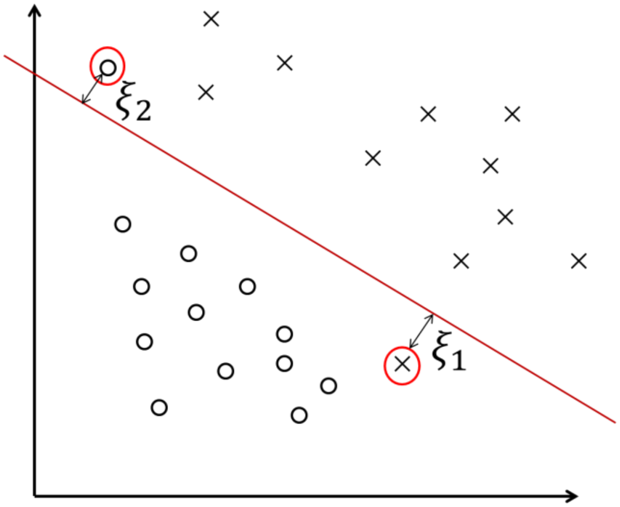


Figure 2.29. Soft Margin SVM showing points of the wrong class either side of the dividing line in red. The slack variables ε are also shown.

The term shown in (2.32) can now be added to the objective function shown in (2.29) and optimization will therefore be a tradeoff between maximizing the margin and minimizing the error where the relative importance of each can be controlled using the error weighting, C.

This implementation of the SVM therefore makes an excellent classifier and can handle high-dimensional, noisy and error-prone training data. There are several downsides to the SVM; not least that both training and classification of new points can be slow. Another issue with the SVM is it can be sensitive to kernel type and kernel parameters. An alternative, simpler approach is the K-Nearest Neighbor (KNN) classifier.

2.4.2 K-Nearest Neighbor

K-Nearest Neighbor works by storing all the training data, then comparing any test points to the training data by looking at the nearest N training points. In this case, Euclidean distance is used and a poll is taken of the K nearest neighbors. The class of the test point is then determined by a vote based on the classes of its neighbors. For example, if a test point is used with K=5, and the 5 nearest neighbors have the following classes {2,1,2,2,2} then the test point will be classified as class 2.

The reasons for using KNN are that it doesn't need training beforehand, it runs quickly and it only has one parameter, K. This makes KNN ideal for initial evaluation of a feature type because classification performance is more closely tied to how good the feature is rather than how well chosen the parameters happen to be. The downside is that the training data has to be stored in memory during execution which could be an issue for large datasets. Finally, the classification performance of KNN is in many cases not considered as good as the SVM.

Chapter 3

System Design

3.1 Overview of System

This system is composed of two major components, the first is the background modeler based on the Gaussian Mixture Model technique which can serve as either an evidence source or a pre-screener for the second component. The second component is the texture based classifier. The two components are then combined in one of two ways. In both cases the GMM acts as a prescreener for the texture analysis. The first method is to use the GMM to prescreen hits and then analyze those hits using texture to get a final confidence value which is projected into UTM space. The purpose of the texture analysis is to modify the hit confidences from the UTM either up or down. The second approach is similar in that it also uses the GMM as a pre-screener to generate hits, but this time with more focus on detection than false alarm rate. The texture analysis is then used to filter out false alarms returning a binary result as to whether or not the hit is genuine based on its texture features. In both methods the texture analysis is trained on a set of hand segmented image patches taken from the IR, (see Section 3.2.). The first idea can be thought of as a softer approach than second. Both methods are tried in an attempt to find the best compromise between detecting all the targets, but keeping false alarms low.

3.1.1 Texture as a Modifier on GMM Confidence

The output confidence from the GMM is thresholded with some low threshold (to reduce number of points to process) and any detections above this low threshold are passed to the texture classifier which returns a simple binary result as to whether or not it believes the object is a target.

The final determination $C_F(x, y)$ is based on the confidence of the GMM hit $C_{GMM}(x, y)$ as well as the decision made by the texture classifier T_o. The combination is done by increasing or decreasing the GMM confidence by a fixed percentage based on the classifier decision. This percentage increase or decrease w_y varies with y-coordinate of the hit as the confidence in the texture measures decreases for objects further down-track, in part because of the low resolution of the imagery. The following describes the calculation of the final output confidence:

$$C_F(x, y) = \begin{cases} \min(1, w_y C_{GMM}(x, y)) & , if\ T_o = 1 \\ \max\left(0, -w_y C_{GMM}(x, y)\right) & , if\ T_o = 0 \end{cases} \tag{3.1}$$

As mentioned w_y is adjusted according to the y-coordinate using

$$w_y = \frac{y}{w_B} \tag{3.2}$$

where w_B is the user-defined base weighting that is then adjusted according to y. It is important to note that in computer vision applications the origin is at the top left of the image and so y increases in the downward direction. Therefore a distant object will have a small y-value and so a small increase in weighting.

Figure 3.1 below shows a block diagram outlining the system:

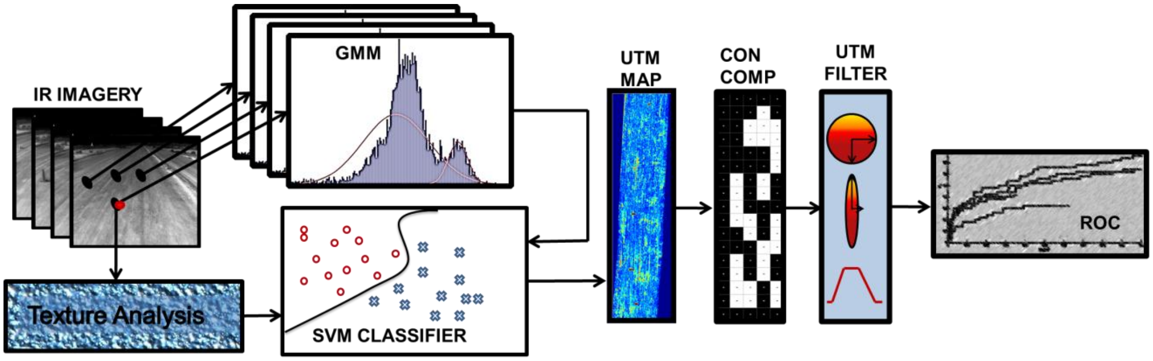


Figure 3.1. System block diagram showing the main components.

After the GMM and classification, each confidence value is projected down into UTM coordinates where a confidence map is constructed. Each cell in the map aggregates multiple detections from the system to create the image. After the confidence map is created it is thresholded, then filtered based on shape as described below.

3.1.2 Texture as an Accept-Reject Method

In this configuration, the GMM is used to generate hits with less regard to the false alarm rate and parameters tuned for good detection rates. The confidence image generated by the GMM is thresholded at a low confidence (~50%) and the connected components found. The centroid of the connected components in (x, y) coordinates is then passed to the texture analyzer. The texture analysis then examines the region around the hit centroids in the original IR imagery and makes a simple binary determination as to whether or not this is a real target or a false alarm. In this method, more trust placed in the GMM due to possible variation in texture at different ground temperatures between runs, any hits found above a certain confidence (i.e. close to 1) will skip the texture step and be projected straight down to UTM before shape-based filtering.

In both cases it must be remembered that the final scoring in UTM space is done on the aggregated hits from the detectors. Because the targets appear in the detection window for up to 15 frames, a single image frame or hit contributes only a small amount to the final shape in UTM and so a few erroneously discarded or allowed hits will not necessarily hurt the detection rate.

3.2 Implementation in Detail

These next few subsections cover the design and implementation in more detail and examine how the components of the system fit together. Changes to the basic algorithms mentioned above will also be covered.

3.2.1 Improvements to original implementation of GMM

The background modeling described in Section 2.23 is closely based on that described in the Stauffer & Grimson paper [8]. Subsequent work such as [10] addressed the theoretical basis of the GMM with special attention given to the update equations and parameters. Experiments were conducted with the following additions or changes to the basic algorithm and evaluated their effect on performance. The changes made to create what here is called UGMM for Updated-GMM are as follows:

- Learning rate is now a function of time or frame number and not fixed as before
- Number of Gaussian components is increased
- Initialization of μ and σ is now dependent on other parameters and settings

Borrowing from [10], a table can be created showing the recommended parameters and the parameters found best on the IR dataset

Table 3.1. Parameter listing for GMM

Symbol	Value From [10]	Value Used with IR	Meaning	Assumption
K	3	5	Number of Gaussian components	
α	0.005	0.015 (fixed)	Learning rate	Based on frame rate if a function of t
λ	2.5	2.5	Standard deviations a pixel is from the mean before calling foreground	
T	0.7	0.75	Cumulative % of components considered background	Based on number of components in theory.
ω_i	0.05	1/K	Initial component weights	Based on number of components in [22]. Weights normalized to sum to 1 after first update
σ_i	30	30	Initial variance of components	Value chosen from the pixel range [0,255] for grayscale.

The implementation used in this thesis differs from the values in [10] in part because they base many of their parameters with the assumption that 3 Gaussian components are used. As the background pixels from a moving platform vary more than those from a fixed platform, the decision was made to increase the number of components to 5 to account for an increase in the number of possible modes. Also, pixels at the edge of the road need to have more than three modes because they can alternate between road, berm, grass or bushes depending on the scene in a given frame. Finally, the percentage of the model considered as included in the background, T, was increased as there are fewer foreground objects and they appear at a lower frequency than the typical fixed-camera traffic footage often used. With the moving camera, the opposite is true as there many modes for the background and lots of variation in pixel values for each location in the image.

3.2.2 UTM Map Filtering

In order to score and test the performance of the algorithm, it is necessary to project the confidence values from the GMM down into UTM space. As the neither the parameters of the camera nor the heading of the vehicle is given, it is necessary to approximate a transform using information from the UTM locations at each frame as well as corresponding pairs of pixels and UTM locations. Heading is inferred using the second difference of the GPS locations and then smoothed with a Hamming window filter. The transformation is learned using an evolutionary algorithm called Covariance Matrix Adaptation-Evolutionary Strategy (CMA-ES) [18]. The idea is that a pool of solutions is generated and this pool is updated iteratively in a manner inspired by evolution. The potential solutions are evaluated by comparing the output from them with the known output from the training data then evaluating the error. The error is then used to update the best candidate solutions [32].

Figure 3.2 below shows a UTM map generated after projection of an entire lane's worth of confidence values down into UTM space. For each map element ("pixel"), the maximum of the set of confidences that map to that location is calculated. The large areas of high confidence (red) correspond with anomalies (known buried target regions), though also visible are several areas of clutter.

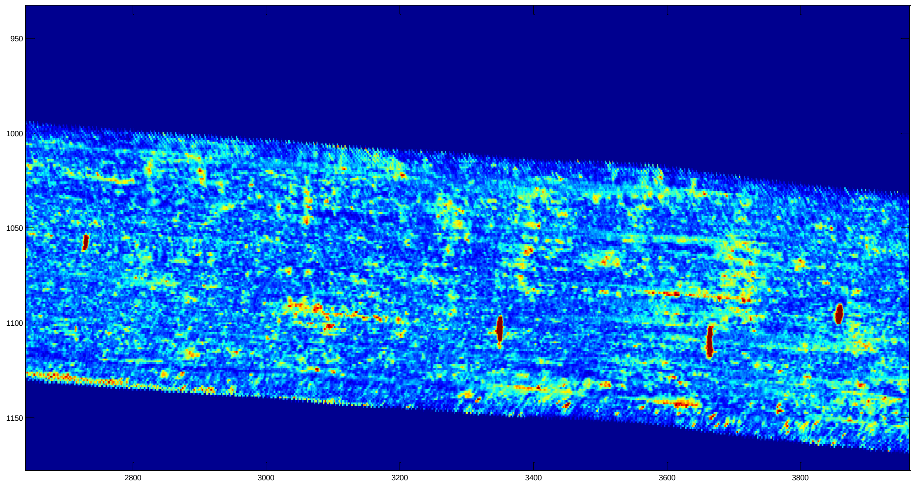


Figure 3.2. UTM map generated by projecting the GMM foreground confidence for each pixel into UTM space. The maximum of the confidences at each map location is calculated. Red areas are higher confidence. The four large red areas in this image correspond to known targets.

Although the GMM outputs can be projected to the UTM coordinate system, there are several sources of error that can affect the accuracy of scoring the performance of the algorithm. First, depending on the platform, as mentioned above, accurate heading information is not always available and has to be inferred. Second, the camera can move independently of the vehicle either due to the bouncing motion over some rough terrain, or for some gimbal-mounted cameras, this can be due to the internal stabilization mechanism of the camera system.

In Figure 3.2, it is shown that the locations of buried targets are associated with high confidence (red) and that they are easily differentiable by visual inspection. However, it is clear targets are not the only areas that show high confidence. Therefore, the application of any basic thresholding technique is likely to generate many false alarms as well. Potential sources for false alarms include areas of uneven road and road edges that appear as local change to the algorithm. From visual inspection of the UTM maps, it is apparent that many of the target objects either appear as narrow ellipses oriented in a cross-track direction, or they look circular. This is in part due to the perspective distortion of the buried objects in image space which tends to make objects appear narrower in the down-track direction. Targets that appear circular tend to be those which show up with poor contrast in the IR. Low contrast targets may be deeply buried, or smaller which would explain their lesser impact on the surface temperature of the soil. What follows is a description of how the shape information from the hits can be exploited in order to further reduce false alarms after the GMM prescreening, texture analysis and projection unto UTM have taken place.

First, the UTM map is thresholded at a given operating (confidence) level to create a binary image. Next, islands are found in the binary image using a connected components algorithm with a constraint enforced on the size of the islands considered. A fuzzy set with a trapezoidal membership function, $\mu_A(n_{CC})$, is used to determine the degree to which the area (cardinality) of the island, n_{CC}, agrees with the expectation of the size of a target (shown in Figure 3.4). A trapezoidal membership function for a property A is defined as follows: given four ordered points along the horizontal axis $a \le b \le c \le d$:

$$\mu_A(x) = \begin{cases} 0, & if\ x \le a\ or\ x \ge d \\ 1, & if\ b \le x \le c \\ \dfrac{\mathrm{x-a}}{\mathrm{b-a}}, & \text{if } a \le x \le b \\ \dfrac{d-x}{d-c}, & \text{if } c \le x \le d \end{cases} \tag{3.3}$$

The parameters of the membership functions were chosen after visually examining the unfiltered UTM maps over several lanes to determine the size and shape ranges of targets. In the case of size, the left side parameterized by a and b were more sensitive as some targets can appear quite small yet there are several areas of clutter that are much bigger than any target.

Next, the shape of the island is examined to determine the eccentricity of the ellipses. This is done by calculating the ratio between the two eigenvalues, λ_{p1} and λ_{p2} (Figure 3.3). First, the covariance matrix is produced for an island and it is subject to eigen-analysis. The measure of the shape's "circularity" is $shape = \mu_B\left(\lambda_{p1}/\lambda_{p2}\right)$. The fuzzy membership function used here is shown in Figure 3.4. The idea is to have maximum membership at the value $\lambda_{p1}/\lambda_{p2} = 1$.

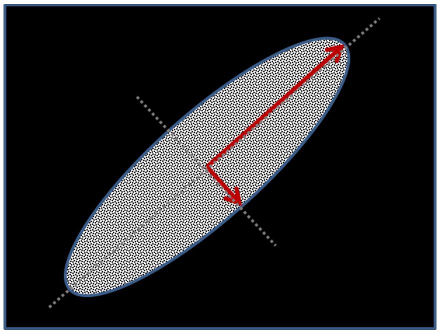


Figure 3.3. Illustration of the eigenvectors for the covariance matrix of an island. The ratio of the eigenvalues can be used to measure shape. The direction of the eigenvector with maximum eigenvalue, i.e. direction of maximal variation can be used to determine primary object orientation.

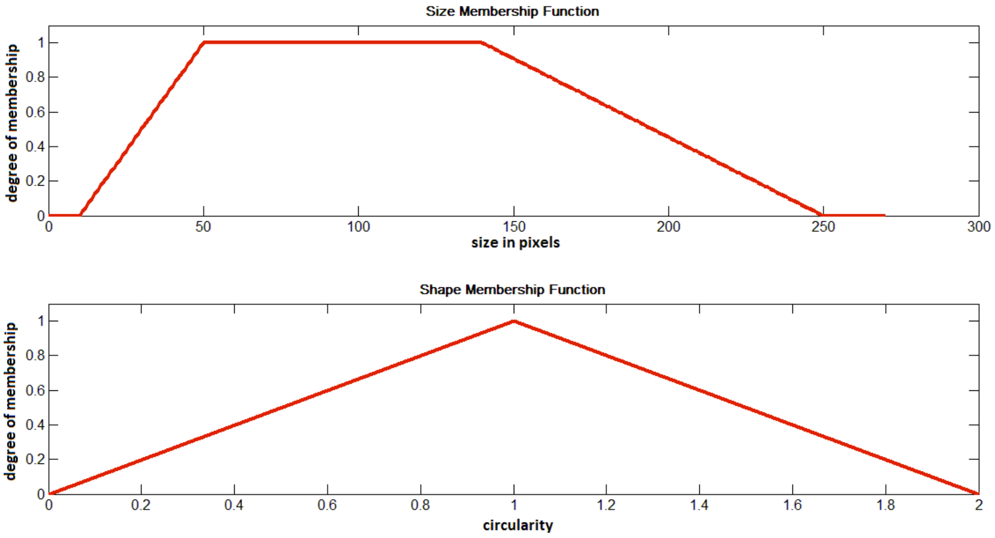


Figure 3.4. Examples of the trapezoidal membership functions used when evaluating size, $\mu_A(n_{CC})$, and shape, $\mu_B(\lambda_{p1}/\lambda_{p2})$, characteristics. The horizontal axes are the domain of the variable, i.e. size of an island in pixels and ratio of the two eigenvalues. The vertical axes are the membership degree.

Due mostly to perspective distortion, it is expected the detected object will primarily take the shape of a narrow ellipse. Following the convention from geometry, here the long and short axes in the ellipse are called the major and minor axes respectively. In order to determine the relative orientation of the objects w.r.t. the lane, the dot product is taken between the eigenvector with the largest eigenvalue, λ_{p1}, and a UTM-space determined vector oriented cross-track. This gives a measure of how much the orientation ($orient$) of the object varies from the cross-track direction. From visual inspection, it can be seen that most targets are oriented with their major axis cross-track, so the larger the dot product, the more certain it is that the island is a target object and not a false alarm.

The degree to which the major orientation of an island matches known cross-track direction is given by:

$$orient = \left|\overrightarrow{majorAxis} \cdot \vec{v}_{p1}\right| \tag{3.4}$$

where $\overrightarrow{majorAxis}$ is the eigenvector with the largest eigenvalue (i.e. the island's primary orientation) and $\vec{v}_{p1}$ is the UTM cross-track direction.

These shape, size and orientation values are combined to give a final confidence that the detected objects are in fact explosive hazards. The confidence of the i^{th} connected component is computed as:

$$P(CC_i) = size * \text{maximum}(orient, shape) \tag{3.5}$$

3. 3 Experiments and Tests

The system described in the previous section is fairly complex and requires the setting of many parameters including the GMM learning rate, initialization, and number of components. The SVM requires that a choice be made about the type of basis function to use and what parameter to use with that function. The texture features all rely on choosing various parameters such as cell size and number of neighborhood pixels, etc. In order to help choose reasonable parameters and pick the best features to use, as well as to ensure proper functioning of the code, a number of tests were conducted before running the final experiments. The first step is to verify each major component separately.

3.3.1 GMM Testing

The GMM was tested first on the simplest case which is some video with a static background and simple foreground changes which consist of a person's hand moving a block or some other object in front of a fixed camera (Figure 3.5). This simple dataset was obtained from [33]. Next, the GMM was tested on the best lane in terms of easily identifiable targets in the IR, and the parameters adjusted to get the best results based on visual inspection of the output. After the parameters had been adjusted for good performance, the GMM was tested as a stand-alone detector without any input from the texture features.

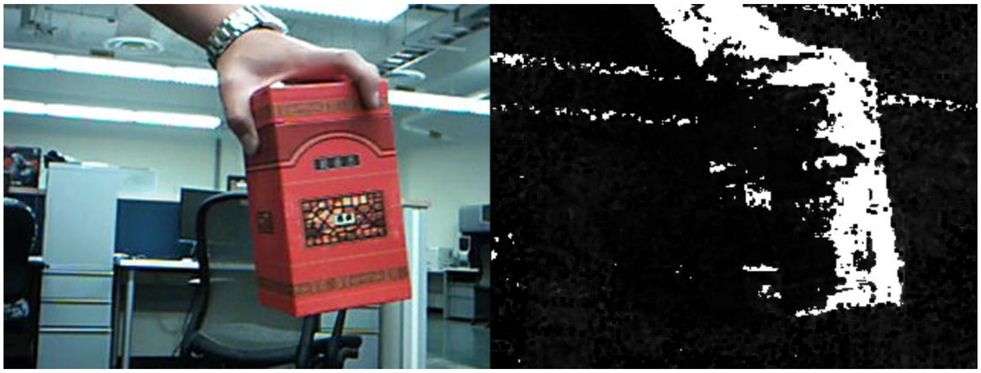


Figure 3.5. GMM run on a simple video of a moving object. In some ways this is more challenging than running on the IR because the box is constantly in frame so the box itself tends to get learned and absorbed into the background if the parameters are not right.

Recall that the method proposed here detects local change or variation as anomalous regions down-track (while a vehicle is moving) in infrared imagery. The algorithm behaves like a high pass filter when viewed from the foreground perspective because only short duration changes are different enough not to be adapted into the background model. The fundamental problem with this approach is that it detects all local changes and not just that those caused by disturbances (physical or thermal) associated with buried objects. This algorithm will include any high frequency changes into the foreground such as quickly changing grades in the berm that lines the road, changes in road surface such as patches of gravel or dirt, channels caused by erosion or rainfall, and various other aspects that account for a less than uniform road surface.

Some of these issues can be addressed by only considering the road surface and not the edges of the road, and by only considering pixels from a window covering the ground at a specific distance in front of the vehicle on which the camera is mounted. This can be achieved by using an image mask and ignoring anything outside our mask area. The masking approach helps to reduce the number of false detections, but does not entirely eliminate the problem. Some of the noise is cleaned up using the shape and orientation filtering, but the danger with this is that not only does much of the clutter remain, but there is a significant risk of rejecting actual hits if the filtering is too aggressive.

Another difficulty is that only buried objects that show a clear high contrast region in the IR image can be detected. Those objects that do not produce high intensity region tend to have too little contrast against the background to be detected. The conjecture is that the return in the IR is related to both the thermal properties of the buried object compared to the local soil properties as well as the burial depth. It is assumed the thermal properties are linked to the metal content of the object, as well as other measures such as relative density with the soil. The presence or absence of moisture in the soil as well as the time of day and local temperature are also possibly factors. For these reasons, GMM is well suited as a method of finding the anomalous regions in the IR as a prescreener as it is not only adaptive, but it detects any short-term deviations from the prevailing background pixel process which is exactly how the buried objects appear.

Algorithm 3.1 . Filtering of GMM outputs

```
FOR thresholds from 0 to 1 DO
    Step 1. Threshold the unfiltered UTM map to get a binary image.
    Step 2. Find the connected components in the binary image.
    Step 3. Find the centroid of the connected components and call these preliminary hits.
    Step 4. Using these points, find the confidence values at the corresponding location in the
    filtered UTM map.
    Step 5. Generate hits and plot the ROC data.
END FOR
```

3.3.2 Analyzing Texture

Next, each texture feature was tested independently to verify proper functioning. Here the Brodatz texture database [34] was used which consists of 64 1024x1024 8-bit image swatches of clear and simple textures (Figure 3.6). The use of sections of the Brodatz was inspired by [14].

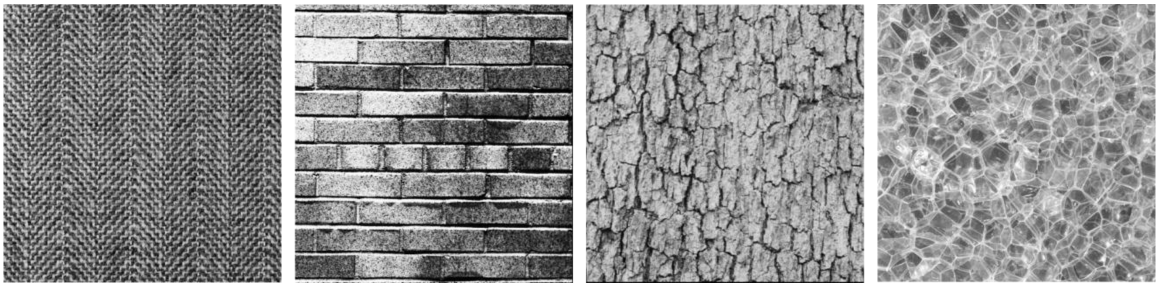


Figure 3.6. Some of the textures found in the Brodatz database.

For each feature type a K-Nearest Neighbor (KNN) classifier was applied to samples taken from each of the Brodatz textures and the performance evaluated (see Table 4.2). Based on these results we were confident that all the code was functional and that the parameter values, though not yet optimized for the IR imagery, were at least in reasonable ranges to produce good results on other grayscale images.

Following the initial testing, the various pieces of the system were again tested, this time the texture feature generators were run on a database of samples actually taken from the IR imagery. The samples were taken at various points in the imagery and consisted of three classes: known targets, road surface and road edge or berm. The same tests above were then run on this database. The database contained the following:

Table 3.2. Number of testing samples for evaluating texture features. The image chips were taken from various positions between 1 meter and 20 meters down-track from the vehicle.

Type	Number 16x16	Number 30x30	Number 50x50
Targets	300	175	163
Road	290	94	201
Berm	28	82	80

Note: at this stage all training and testing samples were taken from the same run to simplify things as much as possible. The KNN was used with a subset of the test images as neighbor data. The results can be found in Chapter 4. For the final combined system the IR segments used to train the classifier were taken from another set of lanes not used for testing during these experiments. This dataset was taken along a stretch of road neighboring the testing lane and contains different buried target objects. This avoids any resubstitution into the texture classifier and so ensures the texture features generalize well at least with the same road type.

The GMM was also run independently on the same testing lane from which the IR image database was created. Here the output of the GMM was filtered using morphological operations and the detection rate was examined. As one of the ultimate functions of the GMM component is to act a as a pre-screener, the main concern at this stage is detection rate and not false alarm rate (FAR), though obviously the lower the FAR the better.

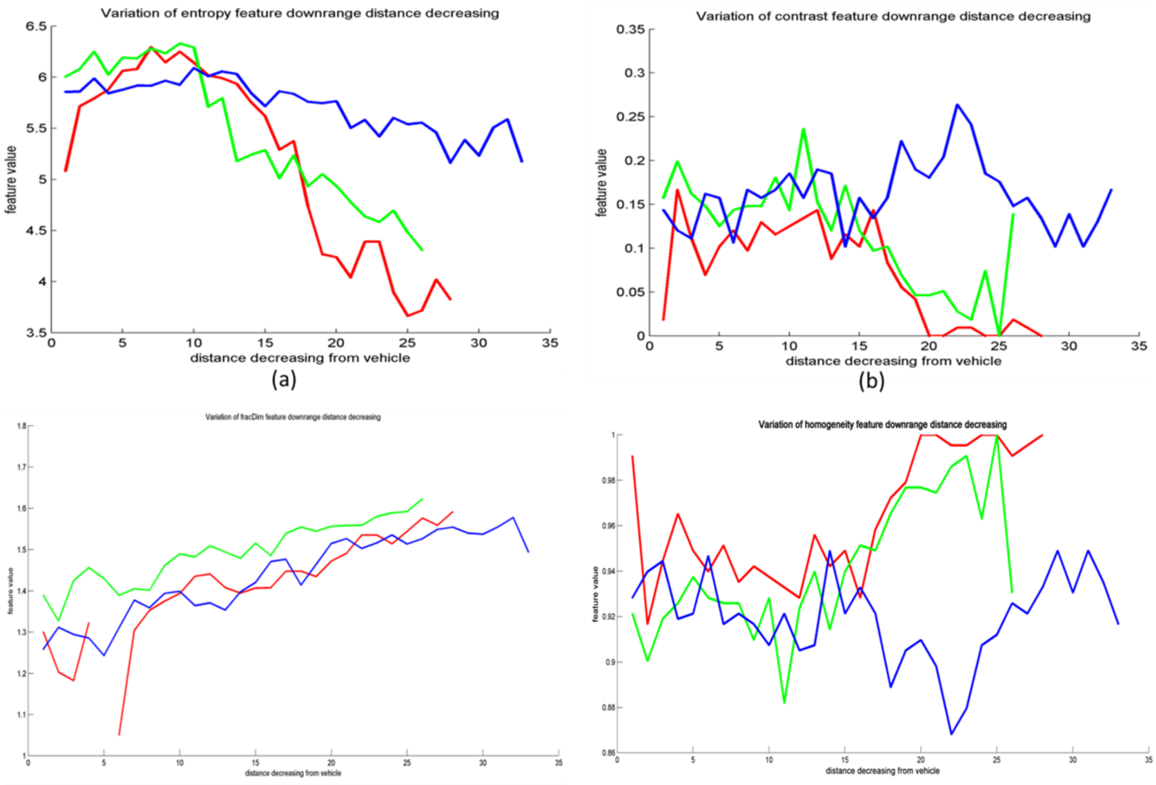
Before the system was put together some final testing was done in an attempt to learn the behavior and characteristics of the features being generated. One main concern is that objects further down-track are nearer the top of the frame and so appear different in shape, size and texture due to perspective. The smaller size of the objects at low y-values (recall, the origin is at the top left for this imagery) means that the number of pixels representing a given object is a lot fewer and so the resolution of any samples taken is also lower. It would be instructive to know what differences there are and how these manifest as the objects approach as the frames progress.

To look at this, several targets were chosen and tracked as they approached the vehicle. Image chips were taken every 2 frames of these objects with care being taken to include the object boundary, but as little background as possible (see Figure 3.7). Finally, image chips were also taken of the road (background) at various distances to see if there were any changes in how the background appears with distance. Figure 3.7 shows the results for entropy, contrast, fractal dimension and homogeneity. Entropy shows a decrease as with the distance from the vehicle (sample 0 is far away, sample 35 is close to the camera) which fits with the interpretation as a measure of disorder. The targets appear more saturated close by which tends to lead to more uniformity in the target regions.

Contrast seems to decrease as the objects get closer for two of the targets, but not for a third. It could be the case that different targets have different properties, which lends some weight to the decision to use multiple classifiers. The fact that contrast drops to zero for two of the targets again implies a uniform area due to saturation of the pixel values close to the vehicle.

Fractal dimension in Figure 3.7.c increases as the targets approach the vehicle. As fractal dimension is supposed to be invariant to scale, it was expected that the value would perhaps differ between targets, but remain constant for a single target down-track. The problem is that the saturation of the objects decreases with distance from the vehicle, and there are probably not enough pixels in the images of targets at far distances to get a meaningful measure of texture. An increase of fractal dimension with decreasing distance implies that the space is being filled more completely which would be expected if saturation is occurring.

Finally, the homogeneity appears to increase for two targets, but not the last. Again, a more homogeneous image would be expected as saturation increases. Why saturation increases is explained when it is considered that the IR camera will pick up more of the emissions from a spot on the ground nearby compared to locations far away. An analogous situation could be how a flashlight looks much brighter close up than it does 25 meters away.



(a) (b) (c) (d)

Figure 3.7. Plots of how features change for the same target as the vehicle approaches. Entropy is shown in (a), contrast is shown in (b), fractal dimension in (c) and homogeneity in (d). The abscissa shows sample number with the first sample being taken furthest from the vehicle, and the last sample being immediately in front of the vehicle. Sample number then is inversely proportional to distance from the vehicle. The different colors represent three different target objects examined.

An interesting observation from these experiments is that for three of the features one target (shown in blue above) appears very different from the others in terms of texture, yet upon visual inspection there is no major difference (see Figure 3.8).

(a) (b) (c)

Figure 3.8. The three targets used in the down-track change investigation. Target (c) showed different behavior than (a) or (b).

After investigating the individual classifiers and the down-track change, the last step was to test the complete system on several runs of the IR data with the SVM replacing the KNN classifier. All test runs were taken of the same lane with the same targets. The difference between runs is the date, time of day and direction of travel. Knowing the time of day and date is important because the appearance of the targets changes with illumination and temperature

Chapter 4

Results

4.1 Examination of Different Texture Features on Brodatz and IR

The first set of experiments to examine the performance of the texture features is shown below in Table 4.1 where the various texture features were tested with the Brodatz database and a KNN classifier. A dataset separate from the testing data was used for training. As the Brodatz images are commonly used in the published research on texture, they make a good benchmark to test the codebase and also give some insight into the effect of the various parameters when used on varied texture types such as grass, brick, gravel, wood, etc.

Table 4.1. Results of classification on Brodatz textures for 16x16 windows.

Algorithm	**Correct Classification Rate**	**Parameters**
LBP Rotational Invariant	99.3%	1x8 mapping
LBP Uniform-2 1x8	50.7%	1x8 mapping K=8
LBP Rotational Invariant and Uniform-2 1x8	99.3%	1x8 mapping K=6
HOG	97.2%	9 bins 3x3 window
Gray-level Co-occurrence stats (4 features)	98.6%	Default MATLAB params
Entropy	97.9%	Default MATLAB params
Fractal Dimension	47.9%	Differential Box Counting

From Table 4.1 it is shown that the best performance is from the LBP followed by the GLCM features. It is not immediately obvious why the fractal dimension feature performed so poorly on these textures (less than chance), it may be because of the small window size of 16x16 pixels. The other poor performer here is the LBP using uniform-2 with no rotational invariance. The addition of rotation invariance makes a difference as the test with the same parameters plus rotation invariance, performs quite well. This result does not carry over to the experiments on IR segments as can be seen in Table 4.2. The difference may be due to the size of the textons in the images. Many of the Brodatz textures tend to have large spheres, stripes or squares in the size range of around 10 to 20 pixels. For this reason neither 16x16 windows, nor 8-pixel neighborhoods may be big enough to capture the structure of the texture. This lack of structure may also be responsible for the poor performance of the fractal dimension measure.

Table 4.2. Results for texture classification on 16x16 pixel IR image segments using the KNN classifier.

Algorithm	**Correct Classification Rate**	**Parameters**
LBP Rotational Invariant	87.6%	1x8 mapping
LBP Uniform-2 1x8	95.2%	1x8 mapping K=10
LBP Rotational Invariant and Uniform-2 1x8	87.1%	1x8 mapping K=10
HOG	92.9%	9 bins 3x3 window
Gray-level Co-occurrence stats (4 features)	85.3%	Default MATLAB parameters
Entropy	85.0%	Default MATLAB parameters
Fractal Dimension	68.1%	Differential Box Counting

Table 4.2 above shows the results of texture feature classification on 16x16 pixel segments from the IR where the two classes are targets and non-targets. Training data and testing data were separate with the targets and background used for training different than those used in testing. All data was taken from the same run. As with the Brodatz textures, the LBP performed the best with a 95.2% correct classification rate. The interesting thing here is that the best classification on the IR is the same parameter set as a poorly performing classification on the Brodatz: uniform-2, 1 pixel radius, 8 pixel neighborhood and no rotation invariance. The fractal dimension performance also increased. This lends credence to the idea that the size of the textons or structures in the imagery is important.

The poor performance of fractal dimension as a feature was surprising as it was expected to perform well on this imagery due to the rough nature of the road surface compared to the more smooth and saturated target images. For this reason a number of additional tests were conducted replacing differential box counting with fractional Brownian motion. The results are shown below.

First, to ensure correct functioning the fBm was run on a test data set generated using a random walk.

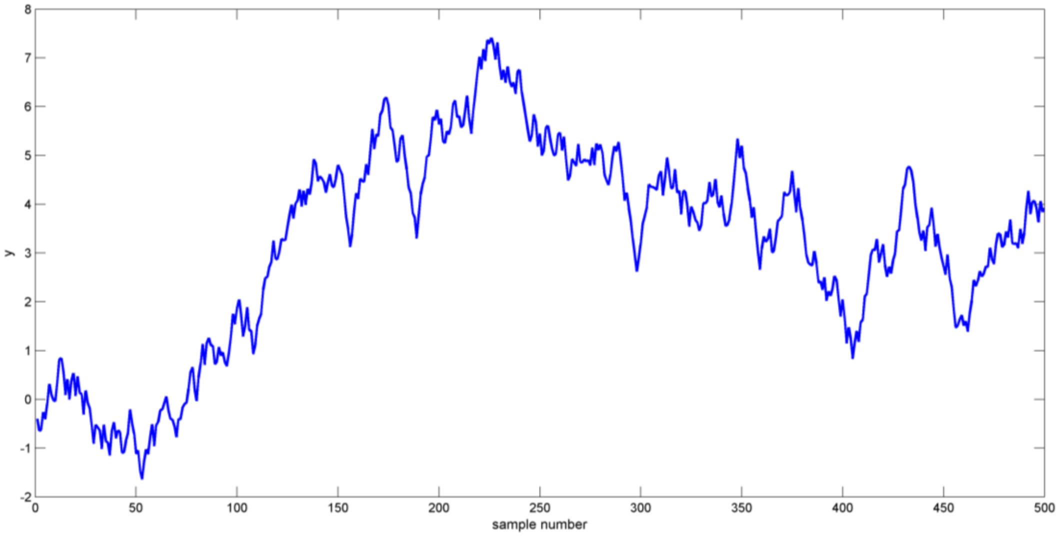


Figure 4.1. Plot of random walk data.

The output from this dataset should show a clear, linear relationship in the log-log plot and this was found to be the case (see Figure 4.2).

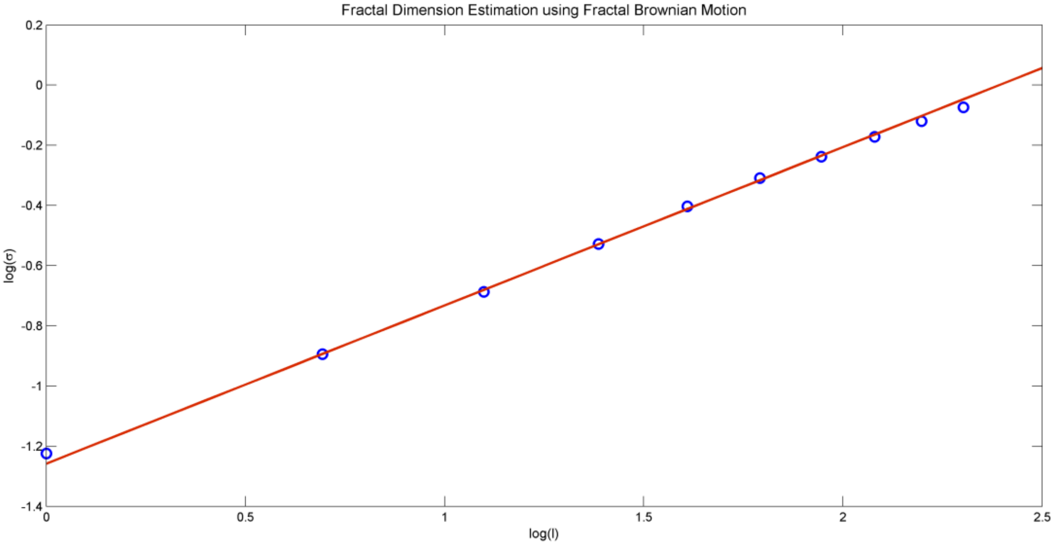


Figure 4.2. Plot of log(σ) vs. log(*l*) showing line of best fit whose slope is proportional to the fractal dimension of the curve.

The fBm was then tested on the 16x16 image chips, but was found to perform poorly; the reason for the poor performance is likely due to the low resolution of the imagery compared to the size of the image chip being used. When larger image chips were tested, the results improved (see Table 4.3).

Table 4.3. Result of experiments with fBm on image chips of various sizes.

Chip Size	Correct Classification Separate Train and Test	Correct Classification Resubstitution	Parameters
16x16	47.9%	50.2%	K=10, max l = 10, trimmed: 2 start, 2 end
30x30	74.6%	95.4	K=10, max l = 10, trimmed: 2 start, 2 end
50x50	81.2%	82.0%	K=10, max l = 10, trimmed: 2 start, 2 end

The results in Table 4.3 demonstrate that image size is important in determining which method of measuring fractal dimension should be used. As the current implementation uses 16x16 image chips, the differential fractal dimension was used in the texture-based classifier. If in future, larger image chips are chosen, it may be advantageous then to use the fractional Brownian motion method over differential Brownian motion.

In summary, the texture based features work well on the grayscale Brodatz images as expected, but what was interesting is the effect that the size of the image window used and the overlap between windows had on the results. As well as demonstrating very clear textures, the Brodatz images are also much higher resolution at 1024x1024 to the 800x400 IR images. Not only that, but the image were taken from above the subject and so the angle at which the texture is viewed does not change. Because of the affect the angle, the IR imagery was not constant in the down-track HOG, LBP and F_D

experiments; there is no simple way of adjusting for this change in angle. The two easiest solutions are to 1: train multiple classifiers at multiple distances or 2: limit the detection region to a single band across the image. The first solution adds to the complexity of the system and is not guaranteed to work as there is significant loss of resolution and therefore texture information as the distance down-track increases. The second solution limits the distance at which we can detect the objects before calling a hit. It is proposed that rather than choose between the two, instead a hybrid approach should be taken. First set a detection region at a reasonable distance from the vehicle and focus most attention on this. Second, have another further band, but give less weight to any objects found here (or not found). In other words do not trust the second, further band as much. A better approach might be to have multiple bands, say one band for every 10 pixels in height, and then fuse the results from each of these bands using a fuzzy integral [35] before projection into UTM space.

4.2 Results for Stand-Alone Detection Using GMM

What follows are the results shown using ROC curves as a means of comparing potential performance under different settings and on different types of inputs. In this case, the algorithm was run using seven different runs taken from the same lane, but at differing dates and times of day. The dates and times of day are listed below. These are important as the number of days since the objects were buried and the temperature at the time of data collection seems to affect the thermal signature of the areas containing buried targets. It is important to make clear that as this is an online, adaptive algorithm there is no separate training data needed. All these lanes are testing data (though you could argue the first few frames of each serve as burn-in for our mixture models).

Table 4.4. List of LWIR runs studied.

Run Name	Date	Time	Number of Buried Targets
Run 1	02/25/10	07:03	50
Run 2	02/25/10	11:46	50
Run 3	03/02/10	06:43	50
Run 4	03/02/10	11:32	50
Run 5	03/04/10	11:53	50
Run 6	02/24/10	14:57	50
Run 7	03/04/10	06:58	50

Figure 4.3 shows preliminary performance of the GMM algorithm plus shape-based filtering with a fixed learning rate of 0.015 for all seven runs with a halo of size 1 meter. The halo size is the distance around a known target location within which a detected object is designated a hit. In other words, if there is a potential hit from the algorithm within 1 meter of a known buried object, then we this is called as a successful detection by the scoring algorithm.

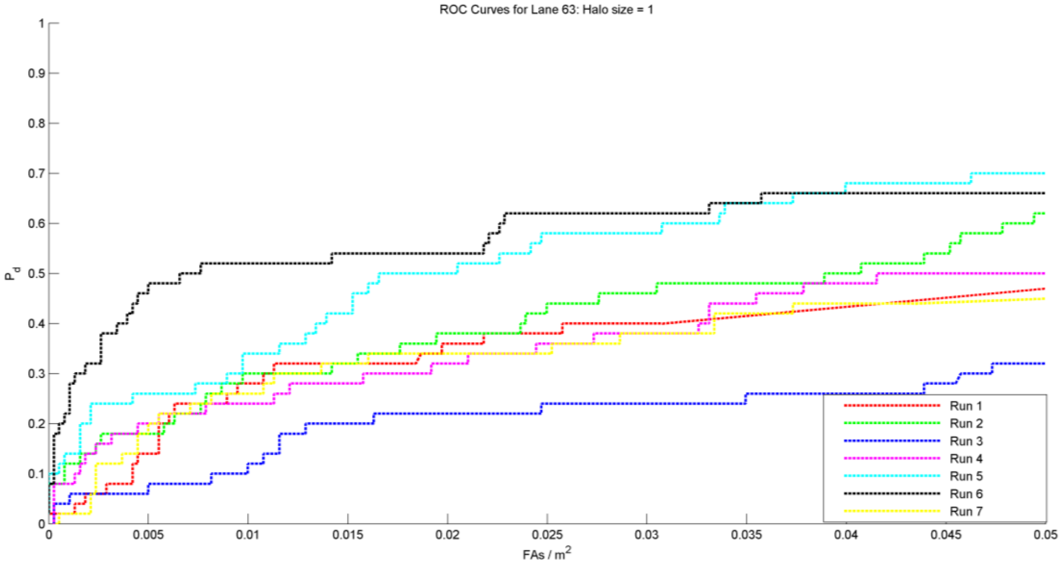


Figure 4.3. ROC curve showing the 7 runs on the same lane with a halo size of 1 meter. Because it is adaptive, our system does not require separate training and testing lanes. Therefore, we don't have to worry about issues like resubstitution bias. We see on average between 50% and 75% of targets.

Figure 4.3 appears to imply the following: first, the morning runs are the hardest; second, the system does not perform well as a direct detector for a fixed set of GMM parameters. The afternoon runs are much better than those earlier in the day. In the morning the observed detection rate is between 28% to 50%. In the afternoon between 60% to 80% detection is observed. The detection rate is fairly close to the limits of the sensor as even visual inspection of the data where the ground truth is known can only detect approximately 85%-90% of the targets.

4.2.1 Improving Morning Run Performance

As the GMM performed so poorly on the morning runs, further analysis was done with the aim of identifying possible reasons and corrections. The main issue is that the targets in the morning are colder than the surroundings and show as black against the background. The problem is that the contrast difference is not as great as it is in the afternoon where the ground above the targets is hotter than the surrounding region. The appearance of targets in the early morning can be seen in Figure 4.4.

In order to improve the appearance and detection of the targets various contrast enhancement techniques were tried. The technique that showed some promise is histogram equalization [16]. Unfortunately, though it does increase the contrast slightly, visually the results in some ways are worse than the raw frame as the background is darkened behind the target (Figure 4.4.b). Because the sky and other large, dark areas of the image can affect the equalization, a local adjustment was done on the detection window only (see Figure 4.4.c). This result is arguably no better than the whole image enhancement, and so this technique was abandoned. Better results may be gained with a more sophisticated contrast or brightness adjustment technique, and this is something that will be investigated in future.

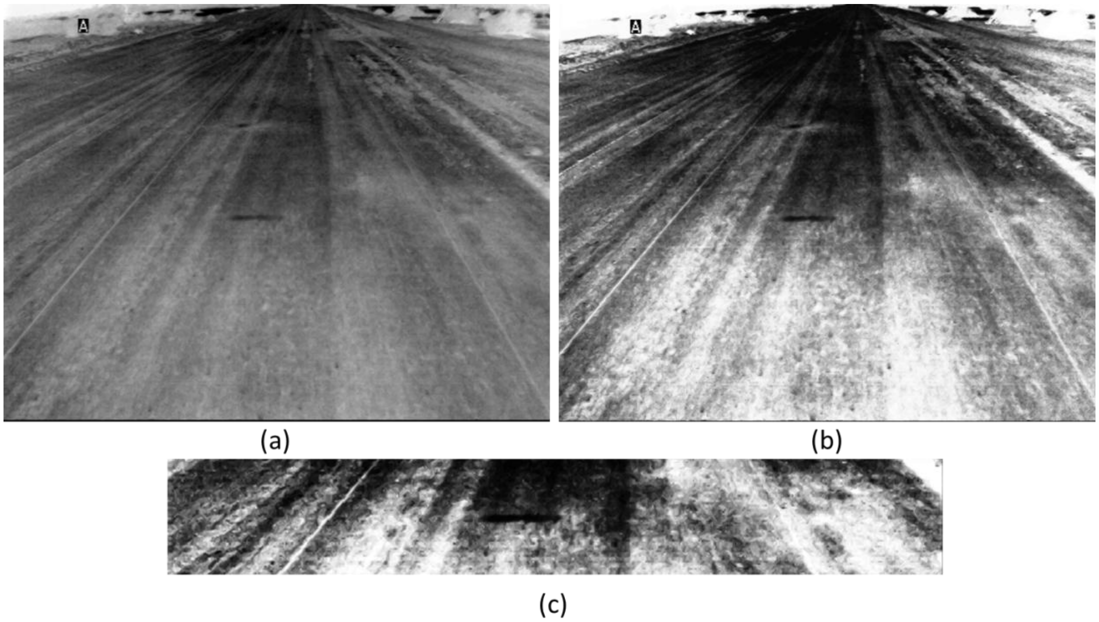


Figure 4.4. Frames from early morning IR run taken around 6 am. The frame in (a) is the original image frame, (b) shows the contrast enhancement using histogram equalization over the whole frame. The image in (c) shows the result of local histogram equalization.

The final test on the morning IR was to invert the images to see if the issue was lack of contrast or some bias in the GMM settings toward a particular gradient direction on the targets. The GMM was rerun on the same data, only this time the image frames were inverted. The results can be seen below in Figure 4.5 and are unexpected. While performance on the morning runs has increased slightly for some runs such as Run 3, it has hurt the performance of the afternoon runs such as Run 6. In theory inverting the image should not change the GMM results because the idea behind the GMM is to detect local change whether it be from dark to light or vice versa.

It would be an interesting future project to examine exactly why the inversion works, and how it could be applied automatically to those lanes where it would improve performance.

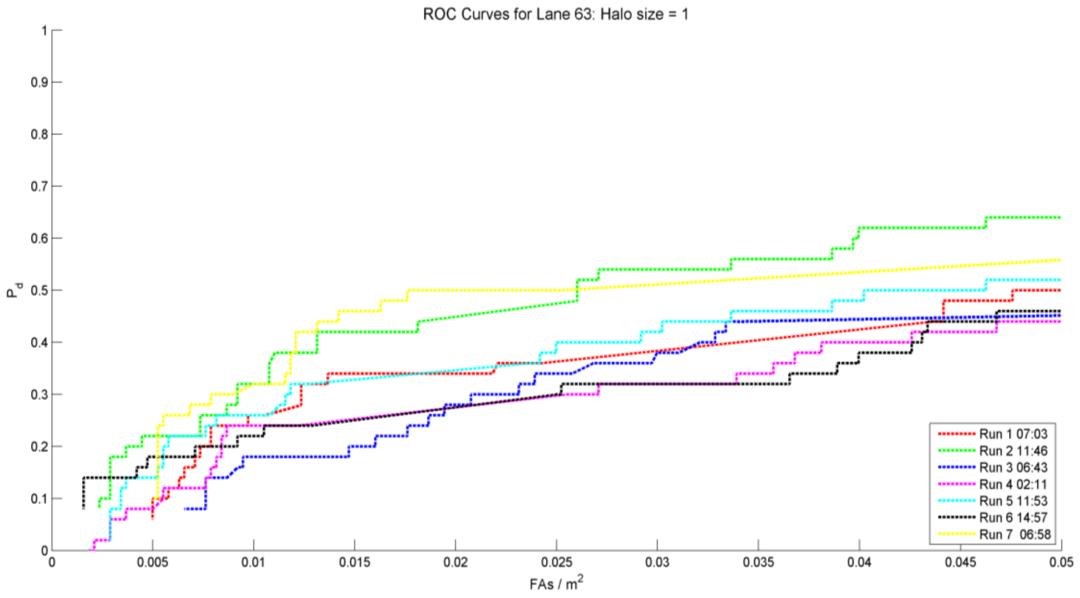


Figure 4.5. ROC curves from GMM alone run on inverted imagery.

4.2.2 Learning Rate Improvement

One of the suggested improvements in [10] for the updated GMM is to allow the learning rate α to decrease with frame number down to a minimum bounding value. Despite several trials with different initializations, the implementation of variable learning rate performed poorly on initial tests with the IR and so a fixed learning rate was used for the rest of the experiments. A number of learning rate values were tried and based on hand-scoring the output images for each parameter setting, a fixed learning rate of 0.015 was chosen for all experiments. The reason for hand scoring is that a human can tell (with the help of the overlaid ground truth) what targets are present among the clutter even when the confidence is very low. An example image for multiple learning rates is shown below.

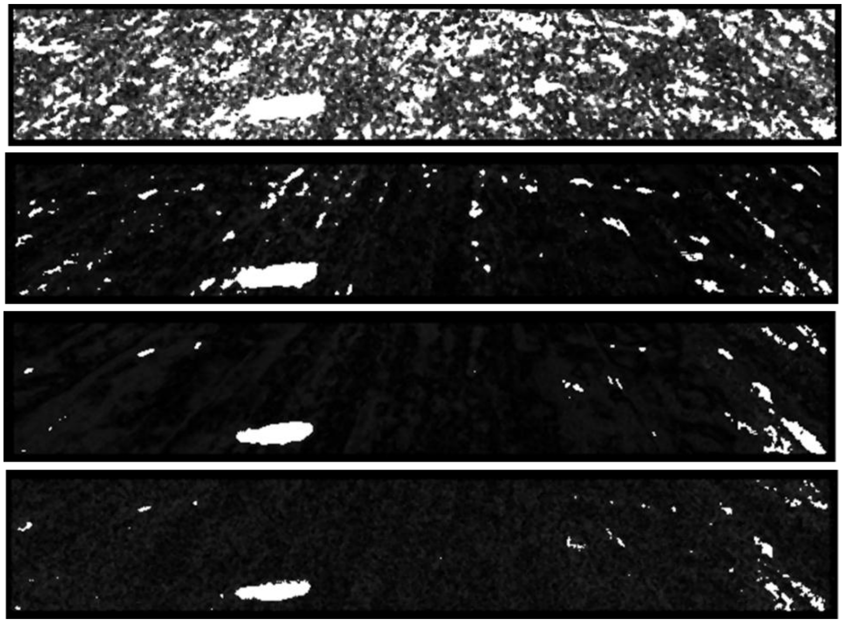


Figure 4.6. Output from GMM alone showing a single target with different learning rate parameter values. The values used are 0.15, 0.015, 0.0015 and 0.00015.

In Figure 4.6 it is clear that as the learning rate decreases, the amount of noise and clutter picked up by the GMM decreases significantly. There is a formula given without derivation in [36] which relates learning rate α, number of components *K* and percentage of model *T* to the minimum time before a value is absorbed into the background. The formula is as follows:

$$t_{min} \geq \frac{1}{\ln(1 - alpha)} ln\left[\frac{K + \lambda - 3}{(K - 2)(\alpha - 1)}\right] \tag{4.1}$$

This equation shows that the number of frames it takes before a new distribution is incorporated into the background decreases as the learning rate increases.

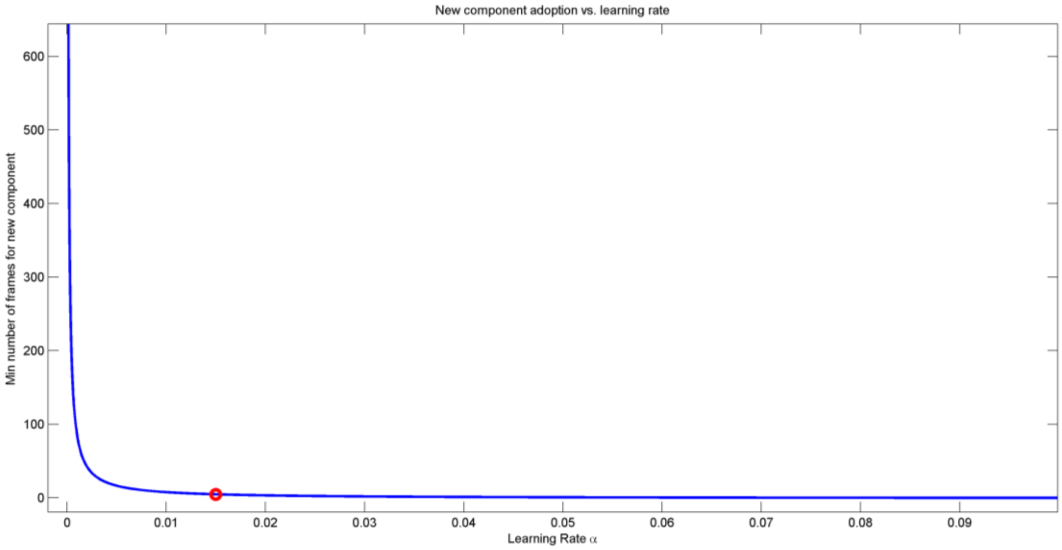


Figure 4.7. Number of frames before new component becomes part of background vs. learning rate. The learning rate used in these tests of 0.015 is shown in red. K=5, T=0.75.

It is proposed that the high noise for large learning rates is likely due to the background model absorbing new components too quickly. When a new component is included as background, it ends up in the part of the model where the cumulative sum of weights T is greater than 0.75. For this to happen, some components with low weights end up in the 25% not considered for pixel matching. If a new pixel no longer matches the model then it becomes foreground, so for the very high learning rates, pixel values are absorbed quickly into the background, but just as quickly pushed out by the absorption of several new values. This may not be such an issue with static camera footage, but with a moving camera the pixel values are constantly shifting.

4.3 Results for GMM as a Prescreener

The figures below compare the performance of the Stauffer & Grimson GMM with a fixed learning rate, but modified updates taken from [10] with no texture filtering (shown in red) to the performance of the GMM as a prescreener with texture filtering (shown in black). In each case, because the texture filtering works so well, there are fewer hits returned by the system. This means there are fewer data points for generating the ROC curves and they cut off at a lower false alarm rate than the GMM alone.

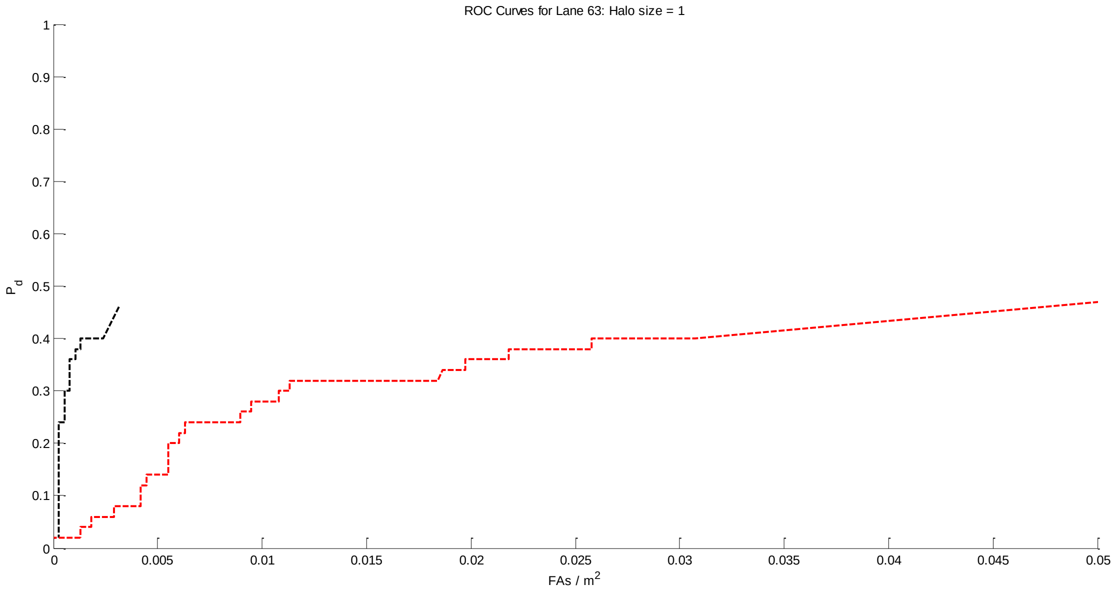


Figure 4.8. Comparison of GMM as a detector (red) and GMM with texture filtering (black) on Run 1.

The ROC above in Figure 4.8 shows a large reduction in the false alarm rate per meter squared (FAs/m^2) whilst allowing an increase in the probability of detection. The increase comes from allowing the thresholding in the shape filtering to be less aggressive (see Section 5.1). The detection rate is still very poor, and this is the case for many of the early morning runs where the contrast between background and targets is poor. This particular run was taken around 7:00 am. See Table 4.4 for a listing of run dates and times.

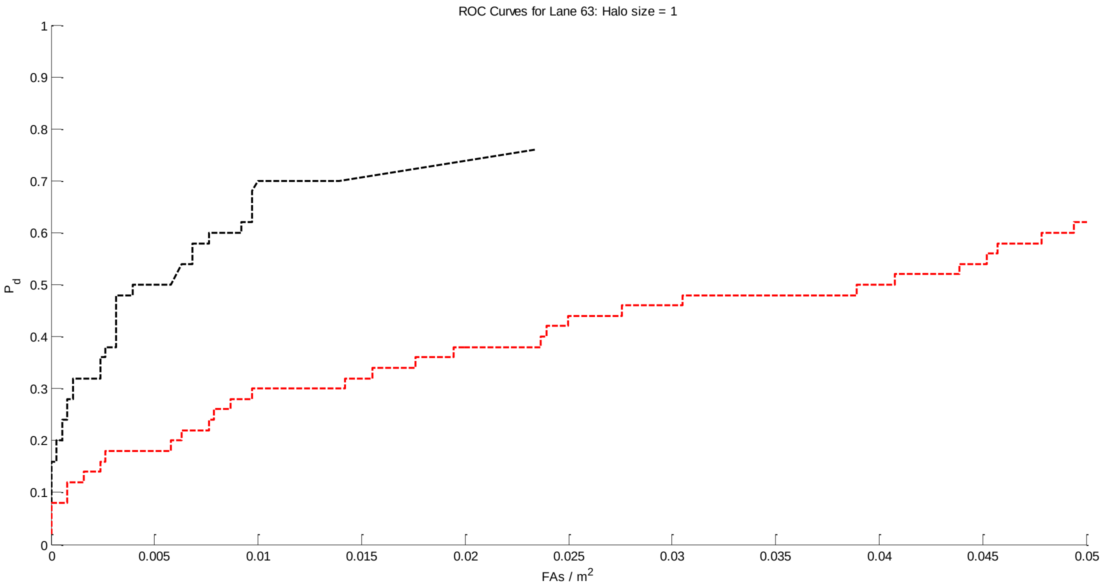


Figure 4.9. Comparison of GMM as a detector (red) and GMM with texture filtering (black) on Run 2.

Figure 4.9 shows a run that took place around noon and the detection rate is much better as expected from the discussion on the basic physics in Chapter 1. Not only is the contrast in the imagery better for this run, but targets show as white against gray, rather than black against gray. The texture filtering has improved the false alarm performance on this run significantly.

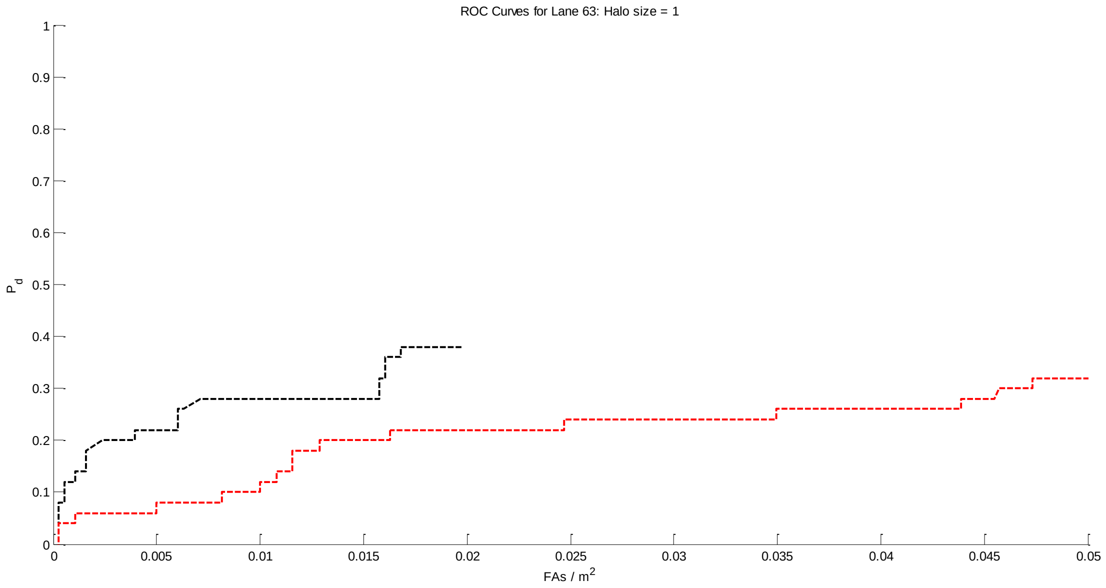


Figure 4.10. Comparison of GMM as a detector (red) and GMM with texture filtering (black) on Run 3.

Above is another run, this time taken around 6:40 am, again as expected the detection performance of the prescreener is poor for early morning runs.

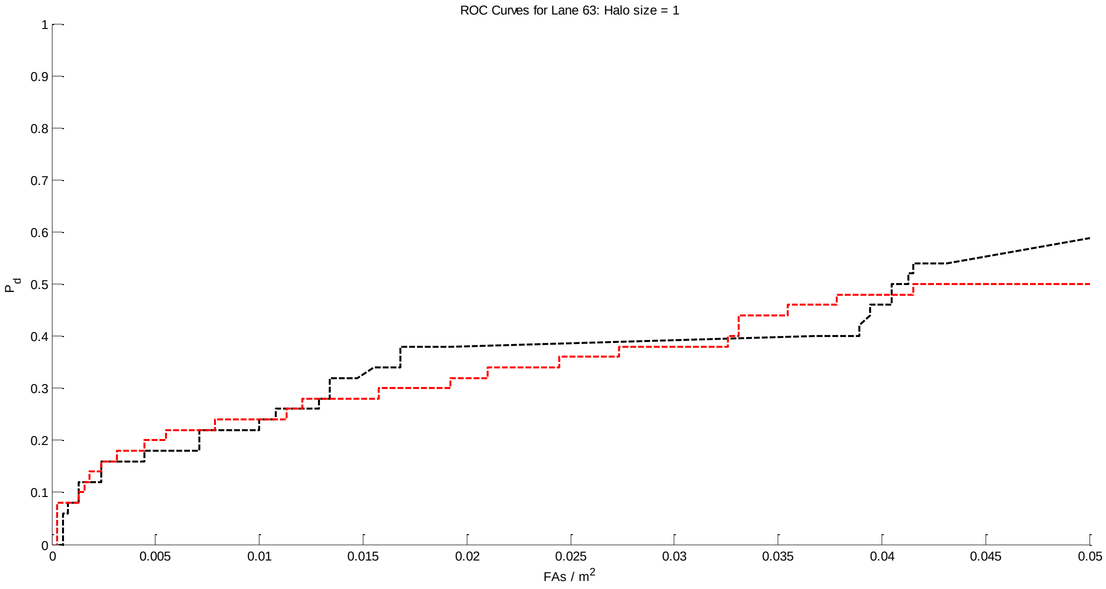


Figure 4.11. Comparison of GMM as a detector (red) and GMM with texture filtering (black) on Run 4.

Figure 4.11 shows a run taken around 11:30 am. As the ground surface heats up, the detection rate rises compared to the earlier run. The difference with this run is that it is midway between early and late for the time runs were taken, the temperature should therefore be between that of the early runs and later runs. The detection performance also seems to be in the middle of the range. The plot is interesting because the performance of the GMM alone and GMM with texture seem to be equivalent

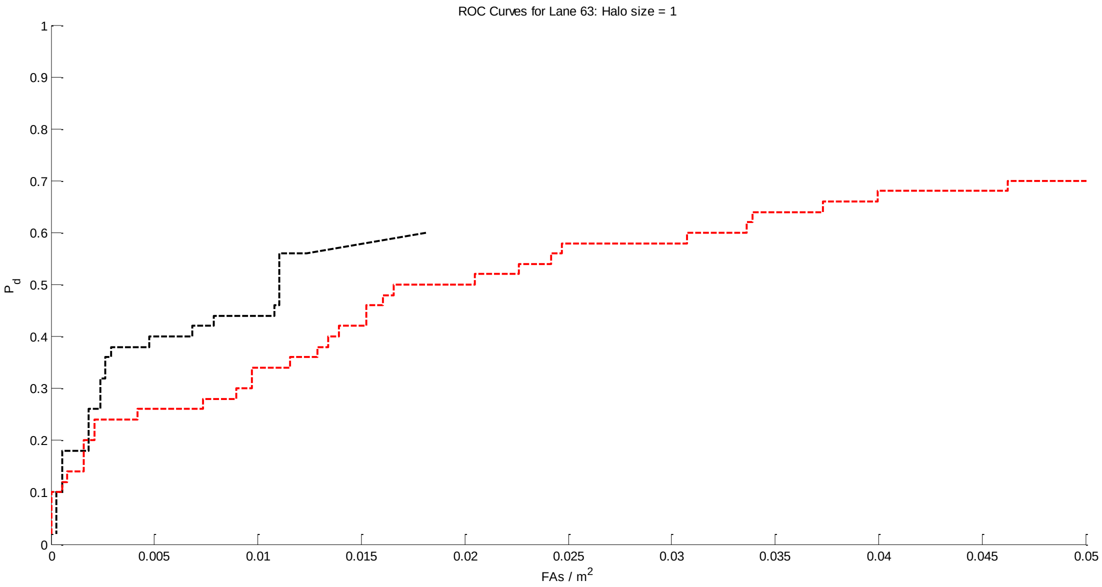


Figure 4.12. Comparison of GMM as a detector (red) and GMM with texture filtering (black) on Run 5.

Figure 4.12 was taken around 11:50 am, and though the false alarms per m^2 have been reduced, the peak detection rate is not as high. This run was taken at a similar time to Figure 4.11, but two days later. This comparison shows that time of day is not the only factor in determining detection performance; though time of day is closely connected, the actual temperature on that day and time is the biggest determining factor. Also, the effect of the presence of moisture or rainfall is not known.

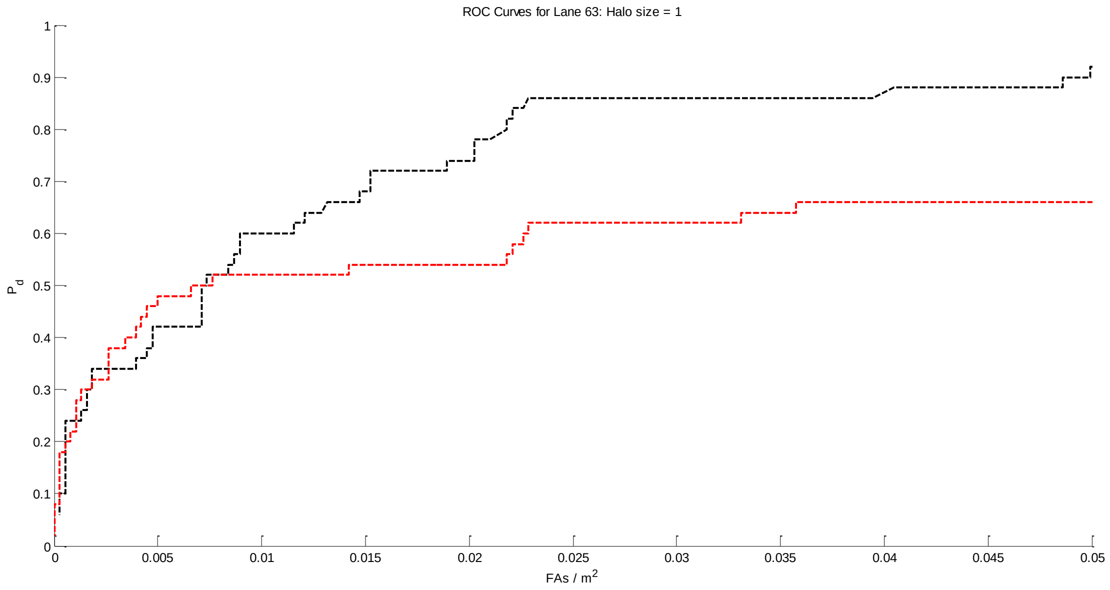


Figure 4.13. Comparison of GMM as a detector (red) and GMM with texture filtering (black) on Run 6.

Figure 4.13 shows the best detection performance of all the runs at 90% with the texture filtered version. The low performance of the GMM alone is due to the prescreener finding most of the targets, but at a very low confidence for several of them. If the confidence threshold of 70% is lowered significantly on the GMM alone version, then the number of false alarms is very large. This is not a problem for the texture filtered version where the GMM is a prescreener. Despite the texture screening, many more hits are returned for this run and this can be seen in the fact the number of data points in the ROC curve matches the GMM alone. The detection performance overall is good which is not surprising as this run is the latest in the day at around 3:00 pm.

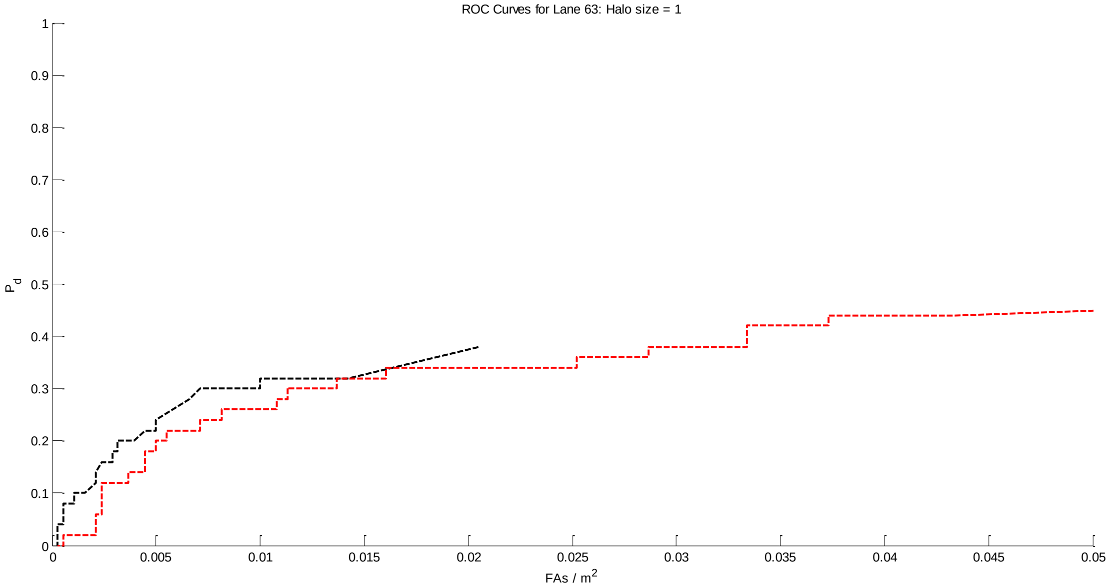


Figure 4.14. Comparison of GMM as a detector (red) and GMM with texture filtering (black) on Run 7.

Figure 4.14 above was taken around 7:00 am, not only is the detection performance poor for this morning run, but the texture filtering only has a small effect on the false alarm rate.

Below, Figure 4.15shows the GMM with texture filtering results for all lanes on one plot (see Figure 4.3 for GMM alone).

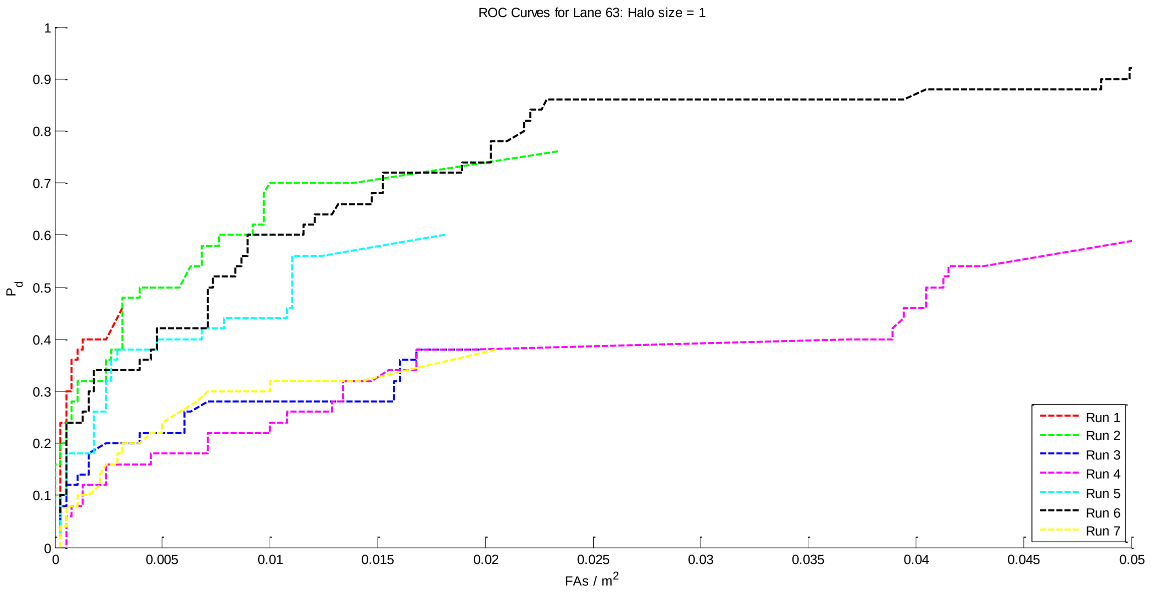


Figure 4.15. ROC curves for all 7 runs using the GMM as a prescreener for texture filtering.

The table below shows more clearly the probability of detection at a false alarm rate of 0.05 for both the GMM alone and the GMM-texture outputs. Just by looking at the probability of detection one can easily pick out the early morning runs.

Table 4.5. Table of Probability of Detection (PD) values at a false alarm rate of 0.05 FA/m^2. Shown are the PD's for GMM alone and GMM-texture.

Run	**GMM Alone PD**	**GMM + Texture PD**
1	48%	46%
2	62%	76%
3	32%	38%
4	50%	58%
5	70%	60%
6	66%	90%
7	44%	38%

The use of texture with the GMM reduces the false alarm rate compared to the GMM alone in every run except Run 4 shown in Figure 4.11. Also, because the texture filtering is so successful at reducing false alarms, it allows a lower threshold on the GMM confidence during the shape-based filtering. The reason for this is that the results from the GMM that have been texture filtered have far fewer false alarms even before texture filtering. The lower threshold means that there is less chance of throwing out genuine, but poorly detected hits. As these hits make it through in the GMM-texture system they increase the PD as can be seen in the ROCs in Figure 4.9 and Figure 4.13, and Table 4.5. This is another example of the benefit of using the Principle of Least Commitment [9] mentioned in Chapter 1.

Chapter 5

Ideas for the Future

Although this thesis has so far focused on IR, detection of buried explosive hazards is not limited to the IR only. There is higher resolution color imagery available which may provide more texture clues, especially when the objects are recently buried and some evidence of disturbed earth is still visible. In most cases from visual inspection of the imagery it is difficult to notice any major texture change and the rough nature of the ground could even add to the false alarm rate without increasing detection. This will be tested in the near future.

One issue at present is the run-time of the GMM algorithm which averages only around 1 frame per second. This set of algorithms fits into what are known as embarrassingly parallel problems as each pixel and its mixture of Gaussians model is independent.

Speedup could therefore be realized by rewriting key parts of the code as a C mex function that could be accomplished using CUDA for C in order that the graphics processing unit could be utilized. There is also the potential for using multi-threading on the host PC at the same time as the graphics device.

There are a number of issues with the data including the lack of accurate heading and camera parameter information. Another factor to consider is that many of the simulated explosive hazards appear to be empty metallic objects. It would perhaps be more accurate if these objects were filled with some substance that mimicked the density and thermal properties of actual explosive compounds. One other factor to consider is the spacing of the targets. For practical purposes 50 targets are buried along the test lanes, yet this density of targets is unlikely in theatre (unless dealing with a mine-field) and therefore the GMM parameters such as learning rate may be adjusted to give better performance in such a situation. If no change has been seen for several miles (instead of the change every few hundred meters), then perhaps any anomalies that do show up will do so more clearly.

Chapter 6

Conclusion

Infrared imagery in the long-wave bands appears to be effective in highlighting buried objects beneath a road. However, the effectiveness of the GMM prescreener (and to some degree the texture classifier) is highly dependent upon the temperature of the soil which in turn is a partly a function of ambient temperature and sunlight incident upon that spot.

More testing in colder climates and multiple times of day is needed in order to fully evaluate the potential of LWIR for detecting buried explosives. The data used for these experiments appeared to show the IR picking up 80% of the objects buried in the test lanes.

Overall, the results on the current IR dataset show some promise, though the detection rate in the GMM is not as high as desired. The results are not as poor as they seem when it is recalled that even a human who knows what the targets look like and with the benefit of the ground-truth can only find around 90% of the targets known to be buried in the lane.

The GMM algorithm shows potential for automating the detection of the targets that are visible in the IR, though it struggles on many lanes due to the variation in the appearance of targets at different times of day. The detection rate could be improved on these lanes by tuning the parameters for different times of day, but this goes against the core idea of using the GMM: it needs no training and is adaptive. A downside to the GMM when used as a standalone detector is the high false alarm rate. Any short-term change in the appearance of the road surface is flagged as a hit. Shape-based filtering ameliorates this problem to some degree, but the results still show a very high number of false alarms. The texture-based filtering of the output of the GMM improves the false alarm rate compared to using the GMM alone. Because the texture filtering works well, the threshold on the GMM confidence output can be lowered to improve the detection rate. This means the texture-based feature classifier allows the performance of the GMM to be weighted toward detection with less concern for false alarm rate at this stage. Texture detection was not used alone on the IR mainly because many of the features are slow to compute, but also because the classifier seems to be better at detecting what is not a target, rather than what is. The benefit of combining the systems then, is that the adaptive GMM takes care of finding the targets in the IR and throwing out most of the rest of the image scene. This means that the range of potential textures to analyze is limited to only those that appear in the GMM output and the run time is significantly decreased compared to using direct detection with texture segmentation.

REFERENCES

[1] Encyclopædia Britannica.. (2011, Mar.) Encyclopædia Britannica Online. [Online]. http://www.britannica.com/EBchecked/topic/72242/bolometer

[2] JJ. Yon, L. Biancardini, E. Mottin, JL. Tissot, and L. Letellier, "Infrared Microbolometer Sensors and Their Application in Automotive Safety," in *Advanced Microsystems for Automotive Applications*. Berlin, Germany: Springer, 2003.

[3] Thành Trung Nguyen, Nho Hào Dinh, Paula Lopez, Frank Cremer, and Hichem Sahli, "Thermal Infrared Identification of Buried Landmines," in *Proc. SPIE*, vol. 5794, Orlando, 2005.

[4] Sen-Ching S. Cheung and Chandrika Kamath, "Robust techniques for background subtraction in urban traffic video," in *SPIE Conference on Visual Communications and Image Processing*, San Jose, CA, 2004.

[5] A. Senior, "Tracking with Probabilistic Appearance Models," in *Proc. ECCV workshop on Performance Evaluation of Tracking*, 2002, pp. 48-55.

[6] N. McFarlane and C. Schofield, "Segmentation and tracking of piglets in images," *Machine Vision and Applications*, vol. 8, no. 3, pp. 187-193, 1995.

[7] Boyoon Jung and Gaurav S. Sukhatme, "Detecting Moving Objects using a Single Camera," in *8th Conference on Intelligent Autonomous Systems*, Amsterdam, The Netherlands, 2004, pp. 980--987.

[8] Chris Stauffer and W.E.L. Grimson, "Adaptive background mixture models for real-time tracking," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1999.*, Fort Collins, CO , USA, 1999.

[9] David Marr, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*, 1st ed. New York: W.H.Freeman & Co, 1982.

[10] P. Wayne Power and Johann A. Schoonees, "Understanding Background Mixture Models for Foreground Segmentation," in *Proceedings Image and Vision Computing New Zealand 2002*, Auckland, NZ, 2002.

[11] Christopher M. Bishop, *Pattern Recognition & Machine Learning*, 1st ed. N.Y., USA: Springer, 2006.

[12] Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*, 2nd ed. N.Y., USA: Wiley, 2001.

[13] Ira Cohen, Qi Tian, Xiang Sean Zhou, and Thomas S. Huang, "Feature Selection Using Principle Feature Analysis," in *ICIP*, 2002.

[14] Timo Ojala, Matti Pietikainen, and Topi Maenpaa, "Multi-scale Grayscale and Rotation Invariant Texture Classification with Local Binary Patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 971-987, July 2002.

[15] Mayang Murni Adnin. Myang's Free Textures. [Online]. http://mayang.com/textures/

[16] David A. Forsyth and Jean Ponce, *Computer Vision A Modern Approach*, 1st ed. N.J., USA: Prentice Hall, 2003.

[17] Navneet Dalal and Bill Triggs, "Histograms of Oriented Gradients for Human Detection," in *International Conference on Computer Vision & Pattern Recognition* , June, 2005, pp. 886-893.

[18] N. Hansen. (2007) The CMA Evolution Strategy: A Tutorial. [Online]. http://www.bionik.tu-berlin.de/user/niko/cmatutorial.pdf

[19] Mryka Hall-Beyer. (2007, Feb.) The GLCM Tutorial. [Online]. http://www.fp.ucalgary.ca/mhallbey/tutorial.htm

[20] The MathWorks, Inc. (2010) Image Processing Toolbox Documentation.

[21] Richard M. Crownover, *Introduction to Fractals and Chaos*, 1st ed. Boston, MA, USA: Jones & Bartlett, 1995.

[22] Phillipe Baveye, Charles W. Boast, Susumu Ogawa, Jean-Yves Parlange, and Tammo Steenhuis, "Influence of image resolution and thresholding on the apparent mass fractal characteristics of preferential flow patterns in field soils.," *Water Resources Research*, vol. 34, no. 11, pp. 2783-2796, November 1998.

[23] Susan S. Chen, James M. Keller, and Richard M. Crownover, "On the Calculation of Fractal Features from Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 10, pp. 1087-1090, October 1993.

[24] F. Moisy. (2008, July) Laboratory FAST, University Paris Sud. Box Counting. [Online]. http://www.fast.u-psud.fr/~moisy/ml/boxcount/html/demo.html

[25] James M. Keller, Richard M. Crownover, and Robert Yu Chen, "Characteristics of Natural Scenes Related to the Fractal Dimension," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 621-627, September

1987.

[26] Nirupam Sarkar and B.B. Chaudhuri, "An Efficient Differential Box-Counting Approach to Compute Fractal Dimension of Image," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 1, pp. 115-120, January 1994.

[27] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, Pittsburgh, Pennsylvania, United States, 1992, pp. 144-152.

[28] Andrew Ng. Stanford Machine Learning. [Online]. http://www.stanford.edu/class/cs229/notes/cs229-notes3.pdf

[29] Tristan Fletcher. (2009, March) University College London. [Online]. http://www.cs.ucl.ac.uk/staff/T.Fletcher/

[30] Ratnanjali Sood and Satish Kumar, "The Effect of Kernal Function on Classification," in *XXXII National Systems Conference, NSC 2008*, 2008.

[31] Christopher J.C. Burgess, "A Tutorial on Support Vector Machines for Pattern Recognition," in *Data Mining and Knowledge Discovery*, Usama Fayyad, Ed. Boston, MA, USA: Kluwer Academic, 1998, ch. 2, pp. 121-167.

[32] Kevin Stone, James M. Keller, K. C. Ho, and P. C. Gader, "On the registration of FLGPR and IR data for the forward-looking landmine detection system and its use in eliminating FLGPR false alarms.," in *Proc. SPIE, Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XIII*, vol. vol. 6953, CID: 695314 (2008), Orlando, FL, 2008.

[33] Changhai Xu. University of Texas at Austin. [Online]. http://www.cs.utexas.edu/~changhai/icra10-datasets/datasets.html

[34] P. Brodatz, *Textures: A Photographic Album for Artists and Designers*. New York, USA: Dover Publications, 1966.

[35] Michel Grabisch, "A New Algorithm for Identifying Fuzzy Measures and Its Application to Pattern Recognition," in *International Joint Conference of the Fourth IEEE International Conference on Fuzzy Systems and The Second International Fuzzy Engineering Symposium., Proceedings of 1995 IEEE International Conference on*, Yokahoma, Japan, 1995, pp. 145-150.

[36] P.L.M. Bouttefroy, A. Bouzerdoum, S. L. Phung, and A. Beghdadi, "On the Analysis of Background Subtraction Techniques Using Gaussian Mixture Models," in *2010 IEEE Acoustics Speech and Signal Processing (ICASSP)*, Dallas, TX, 2010,

pp. 4042 - 4045.

[37] R. M. Haralick, K. Shanmugan, and I. Dinstein, "Textural Features for Image Classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, pp. 610-621, 1973.