THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

Spectrum Sensing
with
Non-local Means

by

David Rubinstein

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Engineering

09/10/2015

Professor Sam Keene, Advisor

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

Dean, School of Engineering - 09/10/2015

Professor Sam Keene - 09/10/2015

Candidate's Thesis Advisor

Acknowledgments

I would like to thank my adviser, Sam Keene for guiding me through this project and helping me narrow down and finding a thesis I'm very proud of. I also thank my family, Adam, Mort, and Sheila, and friends for their support. Thanks to Nicolas Avrutin, Andrew Massimino and Christopher Hong for helping me edit. Finally, I want to thank the Cooper Union for providing me with an excellent education and 6 tuition free years. Without it, I could not be where I am today and do what I currently do.

Abstract

Cognitive Radios depend on the ability to sense unused spectrum in the environment for utilization by a transceiver.There has been little prior work in regards to removing noise (denoising) from detected spectrum before performing spectrum sensing. Instead, most have studied techniques to detect the unused spectrum. In this thesis, two denoising techniques are studied: non-local means and wavelet denoising. Wavelet denoising has been used in the past for spectrum sensing, but only hard thresholding has been studied. This thesis will look at a soft thresholding technique instead in order to improve results. Non-local means is widely used in image processing for denoising, but rarely used in wireless communications. Two variants will be studied tailored to the Rician and noncentral χ^2 distributions. Simulations will show that the soft thresholding wavelet denoising and the variants of non-local means improve energy thresholding performance with non-local means proving to be the superior method.

Contents

List of Figures

1 Introduction

The frequency spectrum is a limited resource and to allow for a reliable communications channel without interference between different users, governments around the world allocate bands of spectrum to different groups, such as telecommunications companies, AM/FM radio station operators, and the military as shown in Figure 1. These owners of spectrum are known as primary users. Allocatable spectrum has become saturated in the modern world, especially in the industrial, scientific, and medical (ISM) bands, which were originally reserved for uses other than telecommunications, but now allow for unlicensed communications. Popular wireless protocols such as WiFi, Bluetooth, etc., and devices that output radio frequency, such as microwave ovens operate in such bands. Meanwhile, in the United States, unlicensed spectrum in use is in high demand, but low supply while the remaining spectrum is licensed to telecommunication providers such as Sprint, Verizon, and AT&T, with the remaining spectrum is allocated to the military as seen in Figure 1. Even though these primary users are licensed large amounts of spectrum, they sometimes do not use all of it depending on how expensive and profitable it is to setup radio base station transceivers in areas like the Rocky Mountains or less-populated, rural areas, as seen in Figure 2. The lack of 100% cell coverage has motivated research for the existence of "secondary users" who can transmit on licensed spectrum that a primary user is not using, allowing for a more efficient use of spectrum.

If secondary users are allowed to exist, then they will need to know when a primary user is transmitting in order to prevent interference with the primary user. Cognitive radios are one solution to this problem. These radios are intelligent in that they can be configured dynamically at the physical layer to use different transmitter and receiver configurations on demand. Therefore, the secondary user could tune the antenna attached to the radio to an unused frequency band. The ability to detect when bands of spectrum are being used is known as spectrum sensing.

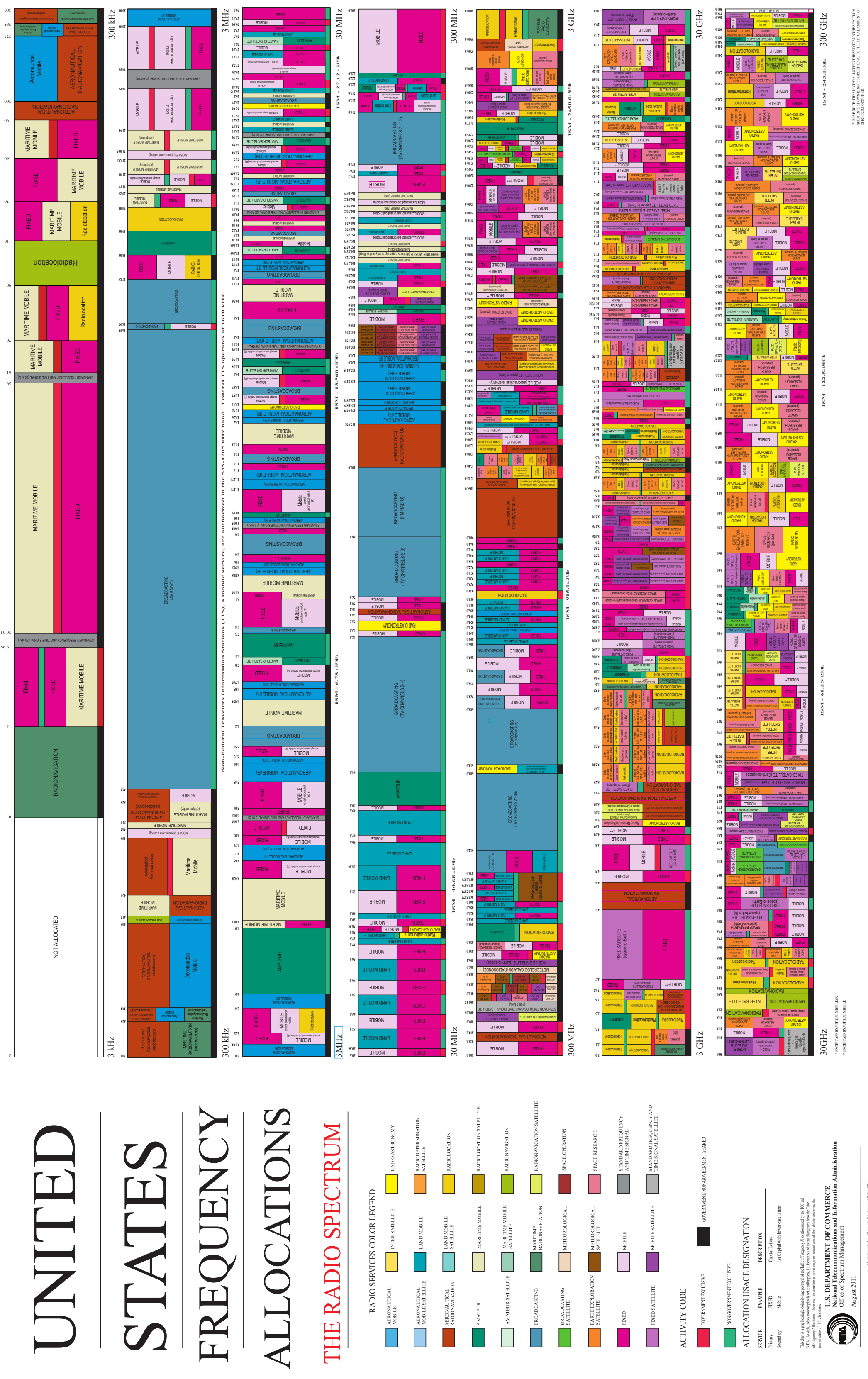


Figure 1: Allocation of spectrum in the United States by the NTIA [1].

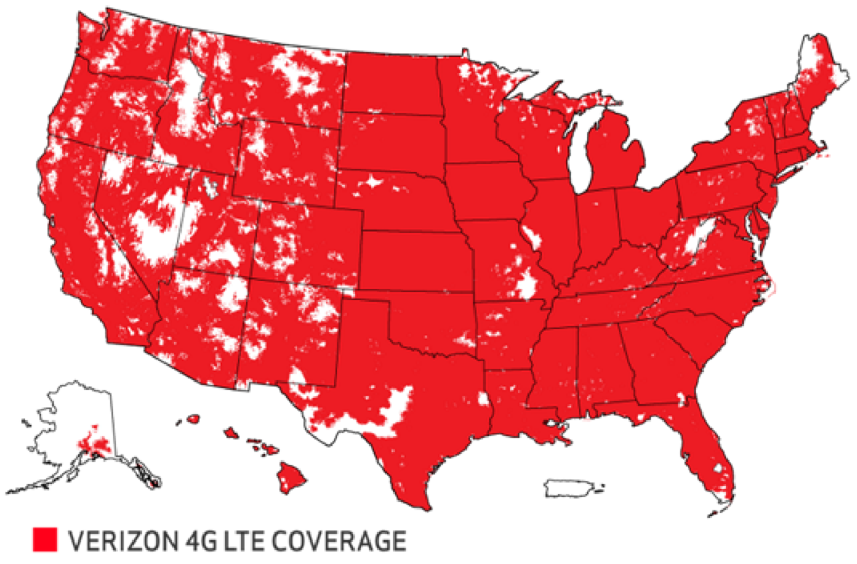


Figure 2: There is dense coverage everywhere but mountainous regions such as the Rocky Mountains and Alaska [2].

Since the early 2000s, there has been an increased interest by both the US government and industry to develop cognitive radios for operation in licensed frequency bands. In 2004, the Federal Communications Commission (FCC) issued a notice of proposed rule-making (NPRM) advocating for the development of wireless regional area networks (WRANs) for cognitive radios in the US [3]. The FCC later freed the analog TV white space bands to provide spectrum for the WRANs [4]. In 2011, IEEE published its first standard for cognitive radio communications, 802.22, for use in WRANs [5]. The initial version used geolocation in order to track what radios are operating, the frequencies of their operation, and their locations. The standard used OFDMA in the PHY-layer and spectrum measurement was implemented in the MAC-layer.

There have been many PHY-layer algorithms created to perform spectrum sensing [6]. These algorithms are used by cognitive radios in order to prevent co-channel interference, which is when two radios are transmitting on the same frequency. One aspect that these algorithms have not studied in depth is denoising, the process of removing noise, which can be used to improve the observed spectrum as a pre-processing stage before performing the detection algorithm. One of the benefits of denoising is that it is not protocol-specific and could be used as a way to improve many spectrum sensing methods and increase the probability that a cognitive radio could detect unused spectrum.

The primary previous attempt using denoising on spectrum relied upon wavelets to mitigate noise by performing a wavelet decomposition, filtering the detail coefficients and then performing the inverse transform [7]. Taking inspiration from image processing, one could try one of the many other denoising methods instead. One of the most popular modern methods is non-local means (NLM) [8]. NLM was originally derived for a 2D image corrupted by additive white Gaussian noise (AWGN). NLM attenuates noise by correcting a target pixel with the mean of many pixels in a large search area weighted by the similarity in the regions surrounding the target pixel and the pixels being used for the averaging. The typical usage of NLM is sub-optimal for communications because received and transmitted signals are complex-valued. Therefore, the spectrogram representing the observed spectrum will come from complex-valued noise. NLM was not derived for complex-valued data since 2D images are real-valued. Assuming the transmitted signal is transmitted through an AWGN channel, if the spectrogram is modified to be real-valued by taking its magnitude squared, the method that is mostly used to analyze a spectrogram, then the noise distribution will be transformed into a non-central chi-squared distribution. If the magnitude is calculated instead, then the noise distribution will be Rician. In order to denoise the data more accurately, NLM should be tailored for the noise distribution of the data.

There has been research in modifying NLM for a Rician distribution [9]. This thesis investigates the performance of NLM and variants of NLM derived for different probability distributions on a spectrogram and how those methods compare to wavelet denoising.

2 Denoising

Noise exists in almost every observation that we make. The severity and type of noise changes depending on the conditions in which the observation is taken and can affect future results. To reduce noise is to attempt to remove corruption from an observation or set of observations. Popular applications of noise reduction or "denoising" include radio receivers, audio, and images. In these applications, the goal is to estimate an observed signal and attempt to approximate the original signal transmitted as well as possible by assuming an underlying noise model, which will be some stochastic process (usually AWGN). Then, taking into consideration the properties and parameters of the noise model, develop an algorithm particular to the model. Often these algorithms have to balance detail and processing power, and will include many user-defined parameters to help with this balance.

2.1 Noise Modelling

The underlying noise model will vary depending on the source and conditions in which the signal is transmitted. Example sources of noise include:

- Thermal, which will change the way electrons flow in silicon.
- The lack of lighting when taking a photo in a dark environment, which will decrease the accuracy of image sensors (cameras), which count the number of photons over the length of the photo's exposure.
- Interference, which will confuse a listener as to what a person is actually saying.
- Lack of precision, which can cause the same repeated task to appear different over many iterations.

These sources of noise will modify a signal in an *additive* or *multiplicative* fashion. Additive means that given a transmitted signal, $x(t)$, received signal, $y(t)$, and noise, $n(t)$, observed at time t, the noise will add to the transmitted signal and the receiving signal will be $y(t) = x(t) + n(t)$. This is often considered easy to process because in addition to properties that will be mentioned later, if the noise is approximated, it can be subtracted out. Multiplicative noise means that the noise is signal-dependent (a function of $x(t)$ and t) and that the receiver will see $y(t) = n(x(t), t)$. It is difficult to remove this type of noise because inverting $n(t)$ (the equivalent of subtracting out noise in the additive case) can be impossible if the noise function is not invertible.. Both methods require estimating the noise distribution in order to improve the quality of the received signal.

2.1.1 Gaussian Distributed

According to the central limit theorem, the mean of a sufficiently large number of iterates of independent random variables will be a standard-normal Gaussian regardless of the underlying distribution [10]. As such, Gaussian (normally distributed) noise is the most popular noise model used because, most data sets with an underlying probability distribution will eventually follow a normal distribution if given enough time for accumulation of samples. The probabilistic density function (pdf) of the normal distribution with mean, μ, and variance, σ^2, that is, $N(\mu, \sigma^2)$ according to [10] is:

$$p(x|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

The mean determines what value x will tend to be and the variance determines the probability that any observation of x will be far away from the mean. The distribution can be visualized as a bell-shaped curve with the peak of the bell being the mean. A particular instance of Gaussian distributed noise is AWGN, which, as the name implies is additive and follows a normal distribution. Thus, in the previous example of additive noise, n would be a random variable following a normal distribution. White means that the power of the noise is the same over all observed information. All of the random variables to be discussed can be derived from a Gaussian random variable. An example of how Gaussian noise can affect a signal is given in Figure 3.

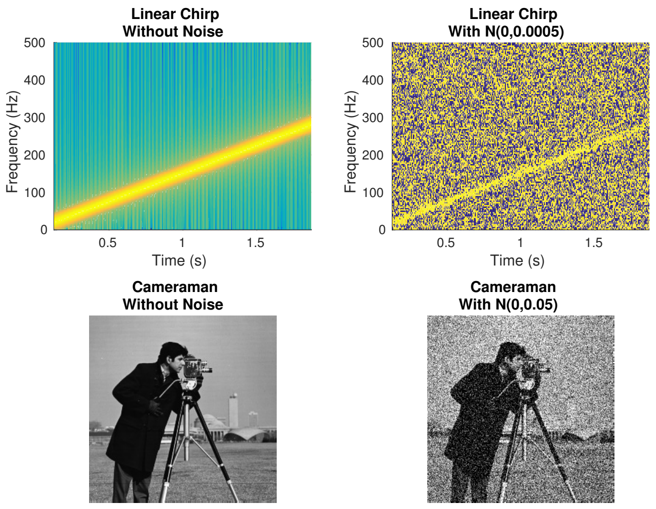


Figure 3: Example of Gaussian noise applied to a linear chirp and cameraman. Note how low the variance of the noise source is.

2.1.2 Rayleigh Distributed

One example of a situation with Rayleigh distributed noise or corruption is the background noise of MRI scans because the background tends to be the magnitude of a complex Gaussian noise [11]. A Rayleigh distribution is the result of taking the norm of two $N(0, \sigma^2)$ variables [10]. Given two uncorrelated Gaussian random variables $X \sim N(0, \sigma^2)$ and $Y \sim N(0, \sigma^2)$, the norm of these two variables is $R = \sqrt{X^2 + Y^2}$. The resulting distribution is referred to as $R = Rayleigh(\sigma)$. The pdf of R according to [10] is:

$$p(x|\sigma) = \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}} \tag{2}$$

The domain of a Rayleigh-distributed random variable is $[0, \infty)$. This distribution can be visualized as top down view at a bell with the bell in the center of your field of vision. This distribution is most often used as multiplicative noise. In Rayleigh-fading, a model for the propagation of a wireless transmission in which there is no line of sight and a large amount of a scattering, the chance of a fade within a channel i.e. attenuation at some frequency within the transmission bandwidth, will occur according to this distribution. The mean of a Rayleigh-distributed random variable is:

$$E[x|\sigma] = \sigma\sqrt{\frac{\pi}{2}} \tag{3}$$

It can be seen that the mean is only dependent on σ, the standard deviation in the Gaussian random variables used to create the Rayleigh random variable. The sole dependence on σ is obvious because the random variables used to create this distribution are zero-mean. Because the mean of the underlying Gaussian distribution must be zero, Rayleigh noise is not an optimal model for the spectrogram because the areas with transmitted energy will change the Gaussian distribution to have a non-zero mean.

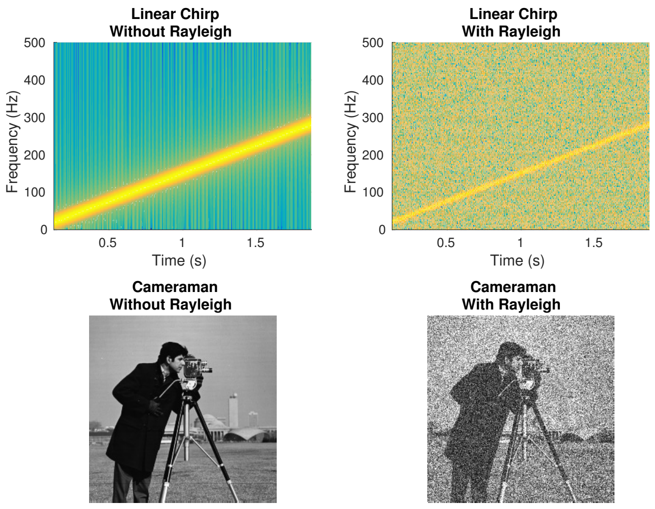


Figure 4: Example of Rayleigh-distributed noise applied to a linear chirp and cameraman.

2.1.3 Rician Distributed

Rician distributed noise occurs when the means used for X and Y in the Rayleigh distribution are not zero [10]. An example of Rician noise is the foreground of an MRI image [11]. The Rician distribution can be considered a generalization of the Rayleigh distribution. An example of this would be trying to approximate the norm of a complex Gaussian random variable with mean $1+j$. In that case, $X \sim N(1, \sigma^2)$ and $Y \sim N(1, \sigma^2)$. Generalizing the example from earlier, a Rician distribution depends on two parameters, ν and σ. X and Y are then modified to be $X \sim N(\nu \cos\theta, \sigma^2)$ and $Y \sim N(\nu \sin\theta, \sigma^2)$ where θ is any real number relating the magnitudes of X and Y. The pdf of a Rice distributed random variable $R \sim Rice(\nu, \sigma)$ according to [10] is:

$$p(x|\nu,\sigma) = \frac{x}{\sigma^2} e^{-\frac{(x^+\nu^2)}{2\sigma^2}} I_0\left(\frac{x\nu}{\sigma^2}\right), \tag{4}$$

with I_0 being the zeroth-ordered modified Bessel function of the first kind. Similar to Rayleigh distributed noise, Rician noise is used as multiplicative noise. Similar to the Rayleigh distribution, this distribution can be visualized as looking down a bell curve, but the center of the bell is a distance ν away from the center, and at an angle, θ, relative to the x-axis. The mean of the Rician distribution is:

$$E[x] = \sigma\sqrt{\frac{\pi}{2}} L_{\frac{1}{2}}\left(-\frac{\nu^2}{2\sigma^2}\right), \tag{5}$$

where $L_{\frac{1}{2}}$ denotes a Laguerre polynomial, which are solutions to Laguerre's equation:

$$xy'' + (1-x)y' + ny = 0. \tag{6}$$

The Laguerre polynomial used in Equation (5) is

$$L_{\frac{1}{2}} = e^{\frac{x}{2}}\left[(1-x)I_0(-\frac{x}{2}) - xI_1(-\frac{x}{2})\right]. \tag{7}$$

Combining Equations (5) and (7), the complete formula for the mean of a Rician random variable is:

$$E[x] = \sigma\sqrt{\frac{\pi}{2}} e^{\frac{-\frac{\nu^2}{2\sigma^2}}{2}} \left[\left(1+\frac{\nu^2}{2\sigma^2}\right) I_0\left(\frac{\frac{\nu^2}{2\sigma^2}}{2}\right) + \frac{\nu^2}{2\sigma^2} I_1\left(\frac{\frac{\nu^2}{2\sigma^2}}{2}\right)\right] \tag{8}$$

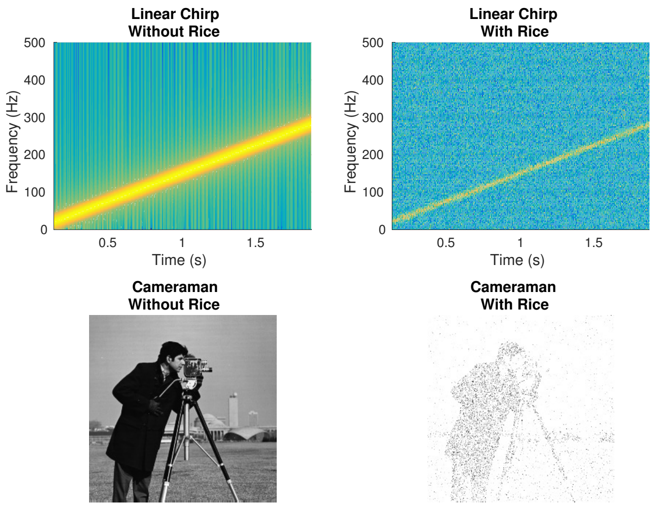


Figure 5: Example of Rice-distributed noise applied to a linear chirp and cameraman.

2.1.4 χ^2 Distributed

A χ^2 random variable with k degrees of freedom is obtained by taking the sum of k standard normal Gaussian random variables (Gaussian distributed with a mean of zero and a variance of one)[10]. If the variables are not standard-normal, they must first be normalized by their means and variances:

$$Y = \sum_{i=1}^{k} \left(\frac{X_i - \mu_i}{\sigma_i} \right)^2 \tag{9}$$

where Y is the chi-squared random variable and $X_i \sim N(\mu_i, \sigma_i^2)$. Chi-squared distributions are famously used in testing variances in populations in what is known as the chi-squared test. The chi-squared test helps determine whether or not many samples observe a Gaussian distribution. The pdf of a χ^2 random variable with k degrees of freedom is:

$$p(x|k) = \begin{cases} \frac{x^{(k/2-1)} e^{-x/2}}{2^{k/2}\Gamma\left(\frac{k}{2}\right)}, & x \geq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Similar to a Rayleigh random variable, this model is good for modeling the background noise, but not the areas with signal power because the mean is non-zero. The solution to finding a suitable distribution for areas with and without signal power will be the noncentral χ^2 distribution. An example of how noncentral chi-squared noise will affect an image is given in Figure 6

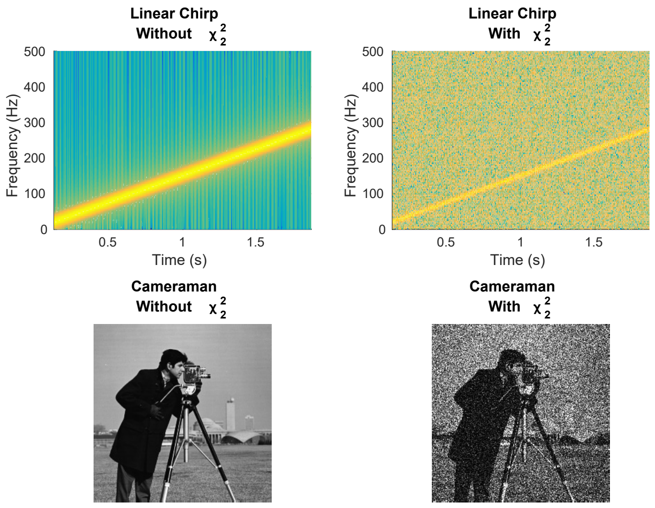


Figure 6: Example of χ^2-distributed noise applied to a linear chirp and cameraman.

2.1.5 Noncentral χ^2 Distributed

The noncentral chi-squared distribution is a generalized version of the chi-squared distribution. It is generated by summing k Gaussian random variables with means μ_i and variances σ_i^2 [7].

$$Y = \sum_{i=1}^{k} \frac{X_i}{\sigma_i}^2 \tag{11}$$

$$\lambda = \sum_{i=1}^{k} \left(\frac{\mu_i}{\sigma_i}\right)^2 \tag{12}$$

In equation (12), λ is known as the noncentrality parameter and represents part of the mean of this distribution. Note that if μ_i is zero for all i, then this distribution is chi-squared with k-degrees of freedom. The pdf of the noncentral chi-squared distribution with k-degrees of freedom and noncentrality parameter λ is:

$$p(x|k,\lambda) = \frac{1}{2} e^{-(x+\lambda)/2} \left(\frac{x}{\lambda}\right)^{k/4-1/2} I_{k/2-1}(\sqrt{\lambda x}). \tag{13}$$

When the transmitted signal is corrupted by zero-mean Gaussian noise, the mean of the Gaussian random variable in the spectrogram becomes the transmitted signal power which changes the distribution of the spectrogram to noncentral chi-squared. This will become very useful when modifying NLM, developed in Section 4. An example of how noncentral χ^2 noise will affect an image is given in Figure 7.

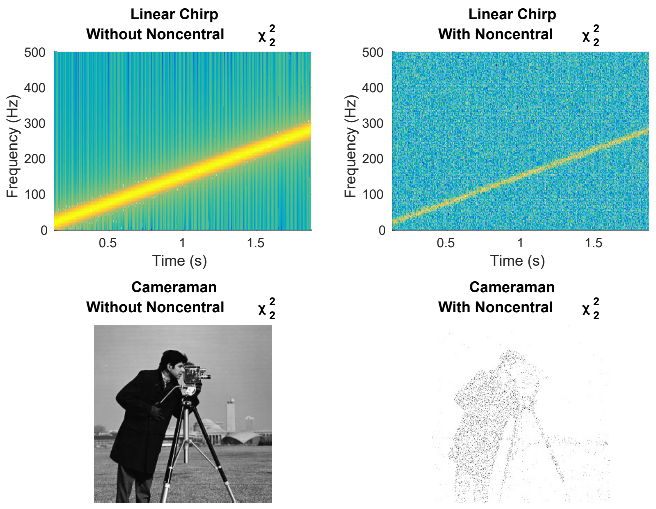


Figure 7: Example of Noncentral χ^2noise applied to a linear chirp and cameraman.

For the purposes of this thesis, this distribution is useful because the magnitude-squared of a complex Gaussian whose real and imaginary parts are standard normal is χ^2 distributed with 2 degrees of freedom. An example of how noncentral chi-squared noise will affect an image is given in Figure 7.

2.2 Image Denoising Methods

This subsection will discuss different types of denoising algorithms that exist and explain how the algorithms function. Most denoising algorithms are designed to use various areas of an image to help denoise a target area. Simpler algorithms utilize a filter using the noise model or exploits AWGN appears as high-frequency noise. A visual comparison of all image denoising methods is given in Figure 16.

2.2.1 Linear Smoothing

One method of reducing noise in an image is to convolve the image with some predetermined filter. The most basic version of this method is to use a low-pass filter to remove high frequency noise in an image. One example of using a low-pass filter is "Gaussian Blur," in which the filter is a 2-dimensional Gaussian with mean zero and some variance specified by the user. The Gaussian acts as a low-pass filter removing the higher frequency noise components of the image. The main problem with using a low-pass filter is that if the original image has a lot of signal power in the high frequencies then much of the detail will be removed. For example, edges are high-frequency components in images since they are sharp transitions between two different areas of an image. Losing high-frequency detail can be very harmful if the end-goal of the algorithm is to separate an image into regions, such as when separating the foreground of an image from the background. Dividing the image into regions is very relevant in spectrum sensing. In spectrum sensing, the foreground is the transmitted signal present in the spectrogram and the lack of activity is the background, the empty areas that the cognitive radio will want to transmit in.

2.2.2 Nonlinear Filters

Instead of having the denoising algorithm operate globally like in the smoothing method, there are algorithms that divide the image into patches (usually of nxn pixels) and perform a filtering operation within that patch. Most of nonlinear filter methods first infer what pixel value best estimates the nxn region of interest based on some statistic of the patch like mean, median, or mode [12]. Each pixel in the patch is then replaced with the estimated value. This heuristic is logical because it can be assumed that for a small enough patch, most pixels in the local area should be approximately the same value. However, this assumption has little merit in areas with lots of high-frequency detail such as edges. Because each patch is being operated on independently, these algorithms are easy to parallelize and implement efficiently. All of these methods are smoothing techniques, so, like the Gaussian Blur method, they may not preserve edges within an image.

2.2.2.1 Median Filtering

Median filtering is a nonlinear method in which each pixel in the patch is replaced with the median value of the pixels within the patch [12]. The algorithm can be summarized as:

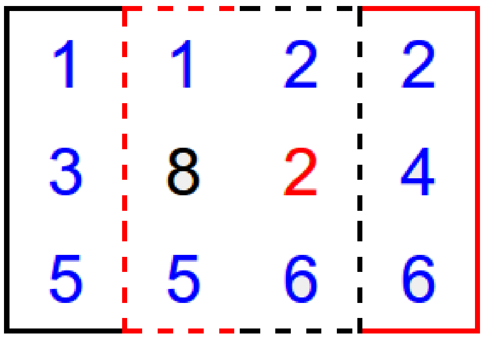
1. Select a patch.
2. Sort pixel within the patch by value.
3. Select the median value within the sorted list.
4. Replace the center pixel in the patch with the median value found in step 3.

Median filtering is effective for removing speckle noise within an image. Speckle noise is Poisson distributed and can be seen when taking a photo in low-light conditions. In some situations, median filtering will preserve edges. For low power of Gaussian noise, median filtering will preserve edges better than mean filtering, which is explained in Section 2.2.2.2 [13]. An example of the median filter operation is demonstrated in Figure 8.

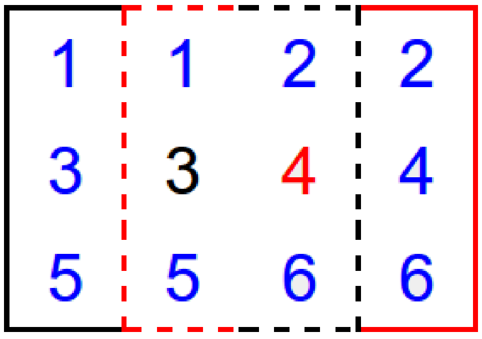
2.2.2.2 Mean Filtering

Instead of taking the median value of the pixels in the patch, mean filtering will replace the center pixel with the mean value of the pixels in the patch [12]. The algorithm can be summarized as:

1. Select a patch.
2. Compute the mean of the pixels in the patch.



(a)



(b)

Figure 8: An example of median filtering where the pixel values are represented by the numbers. (a) contains the original pixel values and (b) contains the new pixel values. The target pixels are denoted by red and black font color. The red and black boxes represent the neighborhoods used to provide the median in the median filtered image for the target pixels shown in (b).

3. Replace the center pixel in the patch with the mean computed in step 2.

Mean filtering is more programmatically efficient than median filtering because it does not require the sorting step. Mean-filtering is mostly used to mitigate the effects of Gaussian noise. The method is very useful for an image corrupted with Gaussian noise because the maximum likelihood estimate of a Gaussian random variable is the mean of that variable. Since it is assumed that all pixels within the patch are from approximately the same Gaussian distribution, the mean of the pixels should be the maximum likelihood estimate for all the pixels in the patch. An example of the mean filter operation is demonstrated in Figure 9.

2.2.2.3 Non-local Means

All methods mentioned previously were "local" methods. Non-local means (NLM) is a non-local method presented in [8] and the basis for many modern image denoising techniques. While a local mean filter presented previously replaces the target pixel with the mean of the pixels in a window surrounding the target, NLM instead uses the similarity between neighborhoods surrounding the target pixel for denoising, as demonstrated in Figure 10. This method has been shown to perform better than the local mean filter, es-

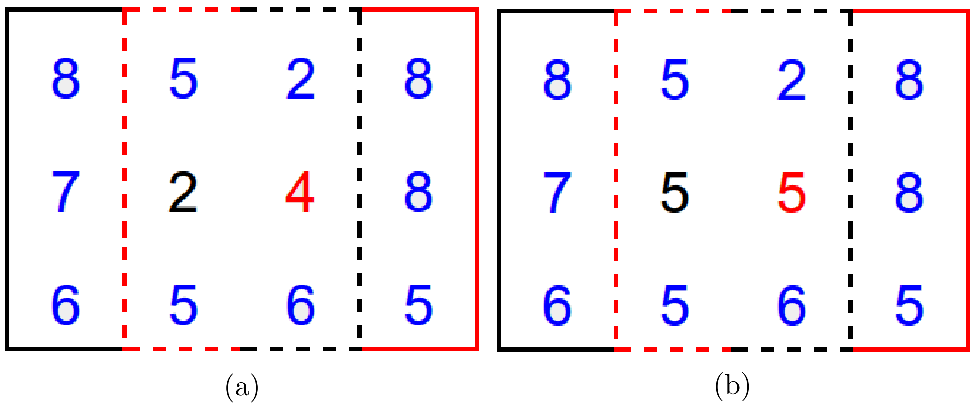


Figure 9: An example of mean filtering where the pixel values are represented by the numbers. (a) contains the original pixel values and (b) contains the new pixel values. The target pixels are denoted by red and black font color. The red and black boxes represent the neighborhoods used to compare the mean in the mean filtered image for the target pixels shown in (b).

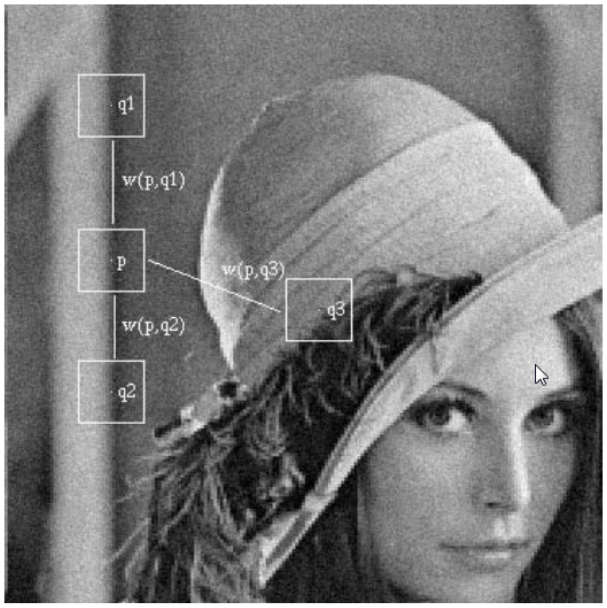


Figure 10: Example of NLM [8]. Pixel p will be averaged with q1-3 and with weights calculated by comparing the regions in the square boxes surrounding each pixel.

pecially in feature preservation. To perform NLM, first calculate weights $w(i,j)$ between target pixel i and comparison pixel j as a function of Euclidean distance:

$$w(i,j) = \frac{1}{Z(i)} e^{\frac{-||v(\mathcal{N}_i)-v(\mathcal{N}_j)||^2_{2,a}}{h^2}}, \text{and} \tag{14}$$

$$Z(i) = \sum_j e^{\frac{-||v(\mathcal{N}_i)-v(\mathcal{N}_j)||^2_{2,a}}{h^2}}. \tag{15}$$

where $Z(i)$ is a normalization factor to ensure that all weights will sum to one, h is a degree of filtering and $\mathcal{N}_i$ represents a neighborhood surrounding pixel i. A Gaussian kernel with standard deviation, a, is applied to the neighborhoods. Note that if two neighborhoods are similar, then the weighting of pixel j will be higher. This is desirable because then it means j has a higher chance of coming from the same distribution as i. Following the calculation of the weights, the target pixel is estimated using:

$$NL[v](i) = E[x] = \sum_{j \in I} w(i,j)v(j), \tag{16}$$

where v represents pixels from the patch before processing and I is size of the search window surrounding pixel i in pixels. Equation (16) is an estimate of the mean of the Gaussian distribution for pixel i, which should be the original value of i before noise was added to the image. These steps are then repeated for all pixels in the image. Another formulation of the method is:

$$NL[v](x) = \frac{1}{Z(i)} \int_\Omega e^{-\frac{(G_a * |v(i+.)-v(j+.)|^2)(0)}{h^2}} v(j)dj, \text{and} \tag{17}$$

$$Z(i) = \int_\Omega e^{-\frac{(G_a * |v(i+.)-v(j+.)|^2)(0)}{h^2}} dj. \tag{18}$$

where G_a represents the Gaussian kernel with standard deviation, a. Upon observation, Equation (17) explains why features are better preserved with this scenario. Because the neighborhoods are being used for similarity, neighborhoods with more features similar to those in the target pixel's neighborhood are likely to be weighted higher, resulting in the features in the patch remaining the same after averaging. In a local mean filter, features are more likely to be blurred because the the target pixel will be using equal weighting for every pixel around it. Thus pixels representing an edge will become the average of the surrounding areas on both sides of the edge, creating a blur.

2.2.3 Wavelet Transforms

The discrete wavelet transform (DWT) and inverse discrete wavelet transform (IDWT) provide variable frequency and temporal information of a signal unlike the STFT which provides the same information, but only at a fixed resolution. The DWT is found by repeating the process of passing a signal through a high-pass and low-pass filter and then downsampling by two. The low-pass filter provides "approximation" coefficients, $Y_{low}[n]$ and the high-pass filter provides "detail" coefficients, $Y_{high}[n]$.

The 1D-DWT is performed by using the method described previously. The transform can be modified into a 2D-DWT by using high-pass and low-pass filters in the horizontal and vertical dimensions, creating four subbands: high-high (Y_{HH}), high-low (Y_{HL}), low-high (Y_{LH}) and low-low (Y_{LL}). Only the LL subband contains approximation coefficients. Each 2D-signal is then downsampled by two. The inverse transform is performed by upsampling by two and then passing the subband images through the appropriate inverse filters. To get higher resolution, successive transforms of the LL subband are taken.

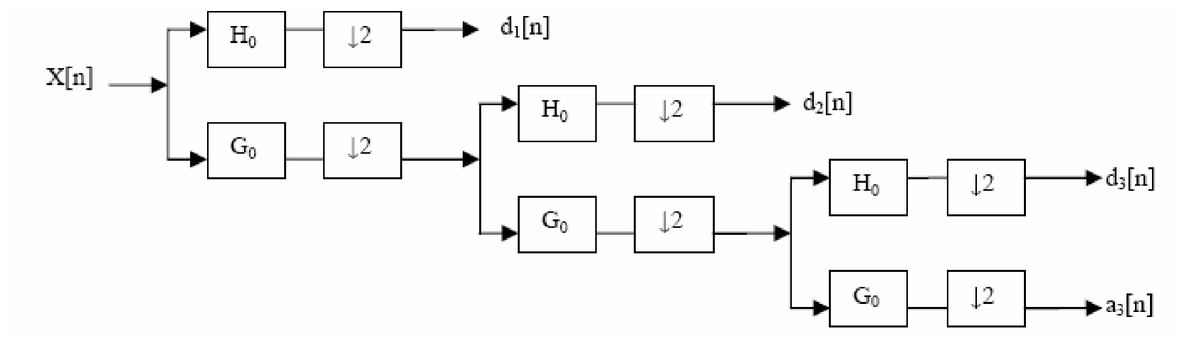


Figure 11: Example of a 1D-DWT [7].

As mentioned previously, white noise primarily manifests as high-frequency noise. Therefore, filtering is usually done only to the detail coefficients. Furthermore, the detail

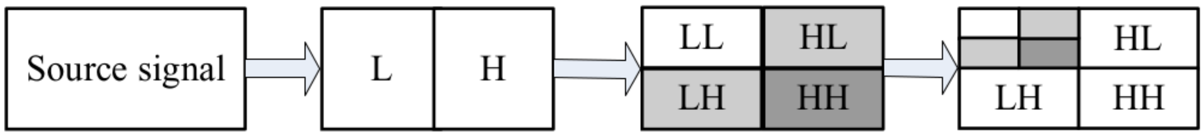


Figure 12: Example of a 2D-DWT [7].

coefficients of a noiseless signal are usually sparse, that is, a large percentage of the detail coefficients are zero. In the transform domain, white noise has lower power than the noiseless signal as demonstrated in Figure 13. An example of performing the DWT on a 2D image is given in Figures 14 and 15. In order to filter out the noise, the signal can be thresholded such that some coefficients are set to zero. There are two variations of thresholding. The first is hard thresholding, also known as *wavelet thresholding* and the second is soft thresholding.

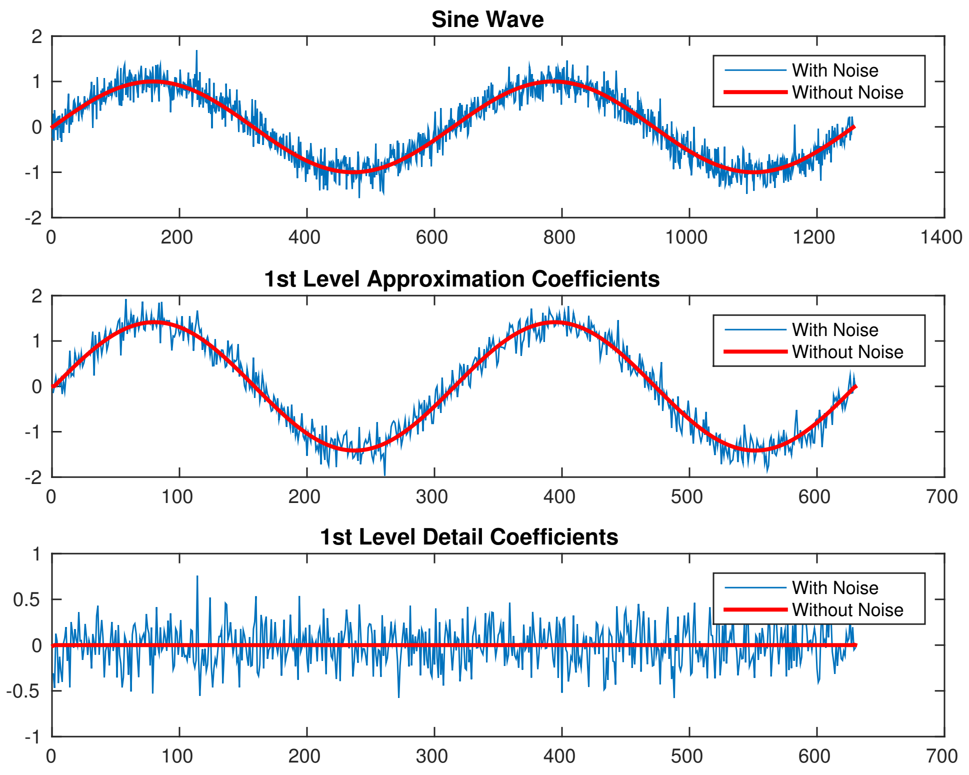


Figure 13: An example of a 1 level 1D-DWT of a sinewave with and without noise using the symlet wavelet. The detail coefficients of the clean sinewave is approximately zero.

Figure 14: The original "Barbara" image before performing the 2D-DWT.

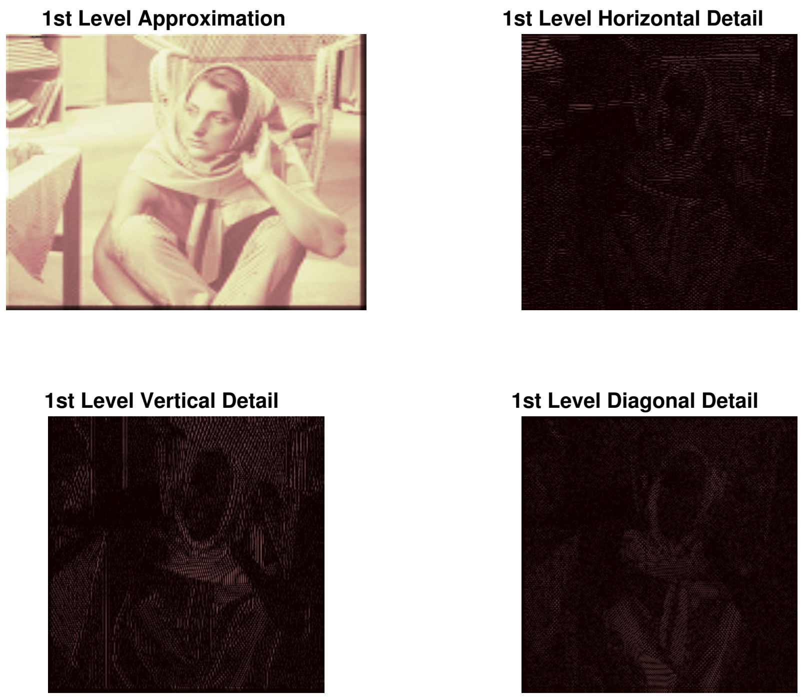

1st Level Approximation
1st Level Horizontal Detail
1st Level Vertical Detail
1st Level Diagonal Detail

Figure 15: An example of a 1 level 2D-DWT of Barbara using the symlet wavelet.

2.2.3.1 Hard Thresholding

There are two variants to hard thresholding. The first method, developed in [14], sets detail coefficients below a predetermined threshold, λ, to zero if the magnitude of the coefficient is lower than the threshold.

$$\eta_{hard}(Y_{high}) = \begin{cases} 0 & |Y_{high}| \leq \lambda \\ Y_{high} & |Y_{high}| > \lambda, \end{cases} \tag{19}$$

where η_{hard} is the thresholding function. The second method, proposed in [7], can be considered a special case of the method in [14]. This method of hard thresholding is performed by setting all detail coefficients to zero. It is equivalent to setting λ to zero in the method developed in Equation (19).

2.2.3.2 Soft Thresholding

Soft thresholding, also known as *wavelet shrinkage* in [14] modifies the thresholding function, η to:

$$\eta_{soft}(Y_{high}) = \begin{cases} 0 & |Y_{high}| \leq \lambda \\ \text{sgn}(Y_{high})(|Y_{high}| - \lambda)_+ & |Y_{high}| > \lambda. \end{cases} \tag{20}$$

Instead of creating the hard boundaries seen in hard thresholding, the soft thresholding function, η_{soft}, creates a smooth transition between the regions near the threshold.

2.2.3.3 Denoising Procedure

The procedure to wavelet denoising is:

1. Take DWT of image
2. Perform hard or soft thresholding on detail coefficients
3. Reconstruct the new, denoised image by performing the IDWT.

To further increase the resolution of the approximation coefficients, repeated DWTs can be taken on successive levels of the transformed approximation coefficients in step 1. The threshold on the detail coefficients is usually a function of the level decomposition it is being performed on according to [14].

The methods developed by Donoho and Johnstone in [14] and [15], VisuShrink and RiskShrink, use hard-thresholding and soft-thresholding, respectively, on all detail coefficients. The soft-thresholding used for RiskShrink was shown to minimize the maximum risk measure:

$$R(\hat{x}, x) = E||\hat{x} - x||_2^2, \tag{21}$$

where $\hat{x}$ represents the estimate of x. The threshold, λ, is designed to create a minimax estimator:

$$R(\hat{x}, x) \leq \Lambda\{\sigma^2 + \sum_{n=0}^{n-1} \min(x^2, \sigma^2)\}, \tag{22}$$

$$\hat{x} = \eta_{soft}(Y, \lambda\sigma), \tag{23}$$

where $\Lambda \leq 2\log(n) + 1$ and $\lambda \leq \sqrt{2\log n}$. Additionally,

$$\Lambda \sim 2\log n, \lambda \sim \sqrt{2\log n}, n \to \infty. \tag{24}$$

For hard-thresholding, it was suggested that $\lambda = \sqrt{2\log n}$. One improvement to VisuShrink is performing the soft-thresholding on intervals of various sizes (interval-dependent denoising) [16]. The detail coefficients are divided up into regions and each region has a separate threshold. The sizes of the intervals are defined by areas of the image with high contrasting variance. For example, one region could be the foreground and another could be the background. The improved soft-thresholding method can be summarized as:

1. Perform 2D-DWT transform a desired number of times

2. Calculate interval locations of the detail coefficients.

3. Apply soft thresholding to the detail coefficients.

4. Perform the 2D-IDWT.

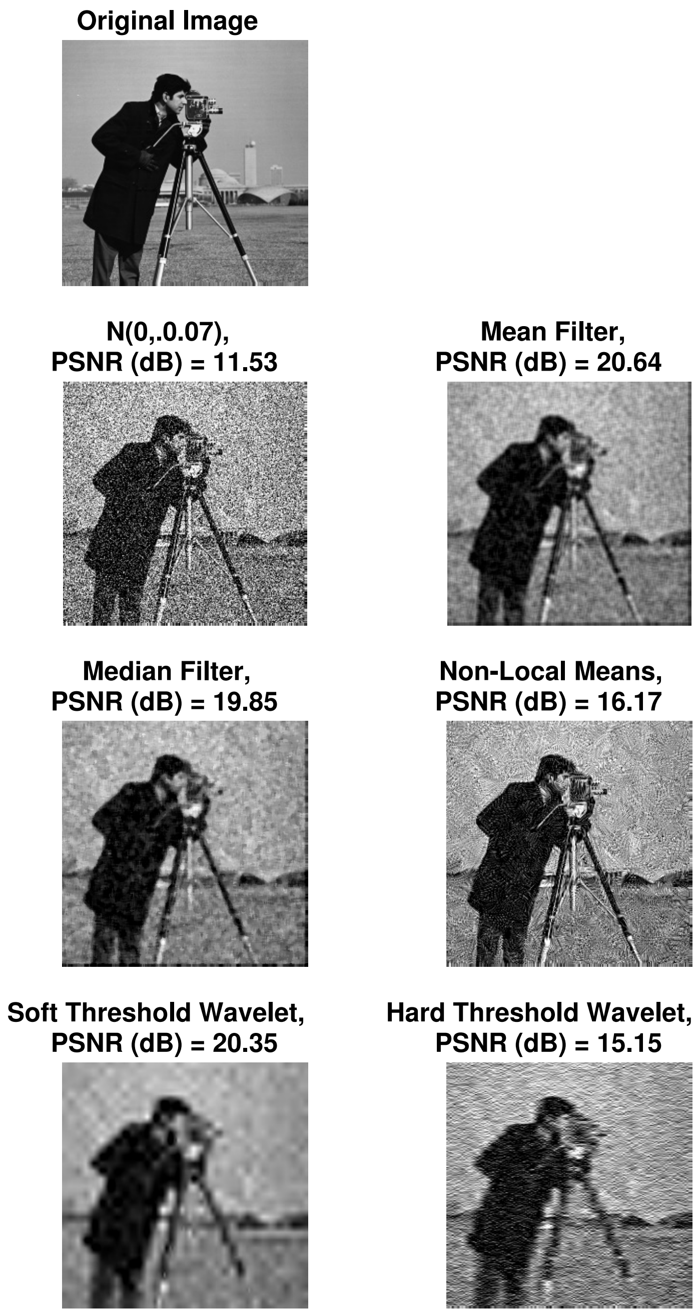


Figure 16: A comparison of different denoising schemes applied to an image of cameraman with AWGN.

3 Cognitive Radio and Spectrum Sensing

Cognitive radio hopes to solve the spectrum allocation problem mentioned in Section 1 by detecting and using unused spectrum from primary users. An example of this situation is given in Figure 17. There are many problems that must be solved for reliable spectrum sensing. If a secondary user interferes with a primary user even unknowingly, the secondary user will will face severe penalties and will be forced to stop transmitting. The problem can be viewed as a variation of the game of Red-Light, Green-Light. The game is played by having many people, representing secondary users, try to approach a person, representing the primary user, who is declared a target, representing the primary user. The secondary users, are a group of people whose job is to tag the target. However, the taggers can only move when the target is not looking. The target declares they are about to look by yelling out "red-light, green-light, one, two, three," at which point the group of taggers freeze. If a tagger is caught moving, then they are sent back to the beginning. The act of the target saying "red-light, green-light one, two, three" represents the time that the secondary user needs to stop transmitting. Similar to someone being penalized for moving while the target is looking at the crowd, the secondary user faces penalties if they are transmitting at the same time as a primary user is transmitting. Instead of being sent back to the beginning, the secondary user would instead face complaints and possibly fines from the FCC.

The reliability of a cognitive radio's spectrum sensing algorithm depends on the operating environment, the monitoring capability and speed of the secondary user's hardware. The secondary user must be able to see active spectrum use in real-time, which is difficult to obtain without hardware assistance from devices such as application-specific integrated circuit (ASICs) or field-programmable gate arrays (FPGAs). There may also be hidden nodes, as portrayed in Figure 18, in the local area, in which a primary user may not be detectable. Reasons for hidden nodes include the shadowing of a far away primary

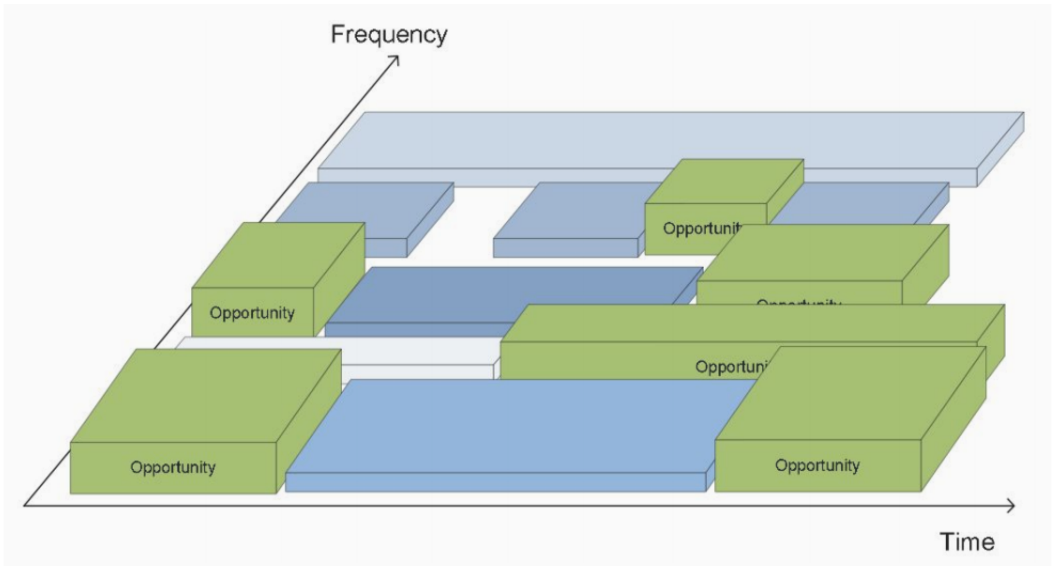


Figure 17: An example of what spectrum sensing would look like according to a cognitive radio. The cognitive radio would see "opportunities" in the frequency spectrum where there are no primary users transmitting [6].

user by a closer user transmitting with more power on an adjacent frequency and fading from the environment due to multipath interference. The radio must also be sensitive enough to detect low-power signals from the primary user. Processing the primary user's signal further decreases the possible time during which the secondary user can transmit. Furthermore, the primary user's signal must be detected immediately when the primary user's transmission begins or previously mentioned penalties may be imposed. These speed requirements mean the detection algorithm should preferably not be computationally complex. A benefit of the secondary user using a non-computationally complex algorithm is a reduction in power consumption which is important if the user is using a battery.

The primary user could be using any possible transmission standard. Most popular standards utilize variants of orthogonal frequency division multiplexing (OFDM) and spread-spectrum (SS). Frequency hopping spread spectrum (FHSS) presents additional difficulty because the spectrum in use is constantly changing with a pattern unpredictable both to the transmitter and receiver. An example of a standard that uses FHSS is

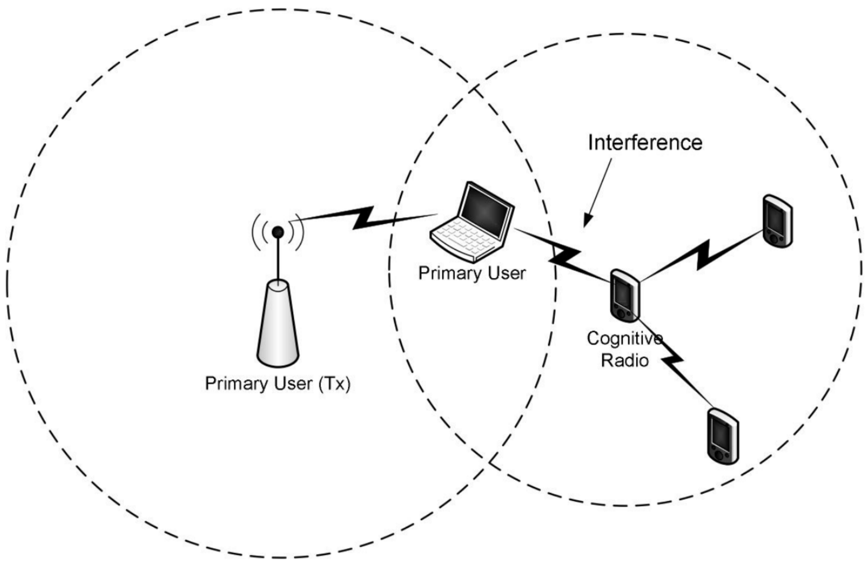


Figure 18: Example of the hidden node problem. The cognitive radio cannot sense that there is a primary user trying to communicate with the laptop because the primary user's transmitter is outside the zone where it can see the transmitter, denoted by the dotted circle [6].

Bluetooth [17]. In addition, LTE's uplink control channel has the option for frequency hopping. Standards using variants of OFDM include 802.11 (WiFi) and LTE's downlink channel [18, 19]. Other examples of spread-spectrum include CDMA and WiFi [20, 18]. Since the wireless protocols in use can vary greatly, it is best that the sensing method is also be flexible.

One way for improving results in spectrum sensing is to use cooperative techniques where multiple cognitive radios in different locations talk to each other about used spectrum. Decision fusion helps solve possible hidden-node problems and has been shown to decrease false alarm rates. Possible ways for multiple radios to cooperate are using the AND, OR, and M-out-of-N methods [6]. In the AND method, a band of spectrum is considered used if all radios agree that the band of spectrum is in use. In the OR rule, a radio will decide whether or not spectrum is free based on whether the sum of all other radios' hypotheses and its own hypotheses exceed a threshold for deciding spectrum usage. The M-out-of-N method is where a band of spectrum is considered in use if M-out-of-N radios believe the radio is in use. The problem with any of these methods is that they require all transmitting radios to be exchanging channel information over an out-of-band or non-wireless channel. One possible drawback is if they are already connected, then there is no need to perform spectrum sensing since infrastructure to handle transmitter communications already exists because they know when each other is transmitting. In addition, the fusion processing adds to the already speed concerns mentioned previously.

3.1 Cognitive Radio Hardware

Progress in cognitive radio has primarily been due to advances in a related field, software defined radio (SDR). These radios implement all facets of the receiver and transmitter chain including mixers, filters, amplifiers, modulators, demodulators, detectors, etc. in software, allowing for a modular radio that can be reconfigured depending on the situation. If the SDR is acting as a transmitter, it transmits the generated signal over some RF front end. If the SDR is acting as a receiver, it first receives the signal via the RF front end, quantizes the received signal and then processes the data in software on a computer or on an on-board chip. SDRs for consumers such as amateurs, university researchers and hobbyists did not exist on the market until the mid-2000s. Since then, the capabilities of SDRs have improved and now have many applications including Basestations (such as OpenBTS), RFID readers, FM radio transmitters/receivers and GPS receivers [21, 22, 23, 24].

One example of an inexpensive SDR meant for consumers is the Universal Software Radio Peripheral (USRP) sold by Ettus Research [25]. Two examples of USRP variations can be seen in Figure 19. The smaller USRP, the N210, is one of the most basic SDRS on the market. It can operate from DC up to 6 GHz and can perform MIMO operations with other USRPs. The larger USRP, the X310, is one of the more advanced SDRs on the market. In addition to the N210's features, it can operate at arbitrary sampling rates and can process data at a higher speed. The USRP connects to a computer through a high-speed connection (usually Gigabit Ethernet), and controlled by host software on a computer. An alternative high-speed connection some SDRs use is USB 3.0. The SDR manufacturer most often provides software drivers in order to help with receiving/transmitting. For example, USRP provides drivers for GNU Radio, LabVIEW, MATLAB and Simulink. In addition, some companies, including Ettus, provide a driver such as the USRP hardware driver (UHD), which can be used in conjunction with C++ or Python programs [26]. The C++ or Python programs can perform the processing within one of the previously mentioned software suites in order to help customize the SDR for the user's intended applications .

Figure 19: On the left is the USRP N210 [27]. On the right is the USRP X310 [28].

3.2 Signal Noise Model

This thesis will assume a signal transmitted through an additive white Gaussian noise channel:

$$y(n) = x(n) + w(n), \tag{25}$$

where $w \sim \mathcal{N}(0, \sigma^2)$ and represents AWGN, $x(n)$ is the transmitted signal and $y(n)$ is the received signal at sample-index, n. Although the Discrete Fourier Transform (DFT) of $y(n)$, $Y(f)$, defined by:

$$Y(f) = \sum_{n=0}^{N-1} y(n) e^{-j2\pi fn/N}, \tag{26}$$

allows for frequency analysis at frequency f, it can only be used to accurately analyze the N samples used to compute the DFT. Thus, a time-frequency analysis method is an important tool for performing any form of spectrum sensing [6]. One example of a time-frequency analysis tool is a spectrogram, a sliding Fourier Transform that yields insight into the structure of a signal and the signal's bandwidth over time. The spectrogram will prove useful later in this thesis for analyzing the observed waveform, $y(n)$, for spectrum sensing. The spectrogram, $S(f, n)$, is the magnitude squared of a short time Fourier transform (STFT), a time-windowed Fourier transform taken at frequency f and time index, n:

$$\begin{aligned} S(f,n) &= |STFT(y(n))|^2 && (27) \\ &= |\sum_{k=0}^{N-1} y(k) exp^{-j2\pi fk/N} \omega(k-n)|^2 && (28) \\ &= |Y(f,n)|^2 && (29) \\ &= |X(f,n) + W(f,n)|^2, && (30) \end{aligned}$$

where $Y(f, n)$, $X(f, n)$, and $W(f, n)$ are, respectively, the STFTs of $y(n)$, $x(n)$, and $w(n)$ performed at sample n and $\omega(n)$ represents the windowing function. Since the Fourier transform is an orthogonal transform, the noise of W is the same as $w(n)$, $\mathcal{N}(0, \sigma^2)$. If Equation (30) is normalized by the variance, the noise will be noncentral χ^2 distributed with two degrees of freedom because the squared magnitude was taken by adding the squares of the real and imaginary components of Y with a mean proportional to the power in the transmitted signal. An example of a spectrogram of a chirp is given in Figure 20. If the squaring is not performed in Equation (30), then the noise will be Rician distributed with parameters ν and σ, where ν is proportional to the power of the transmitted signal and σ is the standard deviation of the Gaussian noise. The mean of a Rician distribution is difficult to estimate because it depends on knowledge of both ν and σ and the ability to invert a Laguerre polynomial. The second moment however, is easily invertible, and will be used instead for estimation.

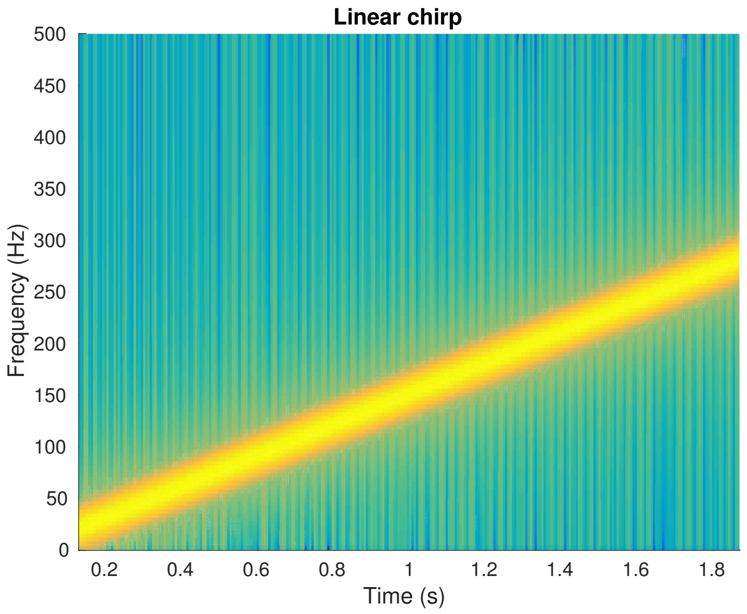


Figure 20: An example of a spectrogram showing a linear chirp.

3.3 Spectrum Sensing Methods

Spectrum sensing methods can be divided into two groups: blind and non-blind. Blind techniques will not use any features of a well-known signal structure in the sensing algorithm. Non-blind techniques will use some features that exist in the signal structure. Non-blind techniques are harder to implement and are more computationally complex because they require possible decoding of the signal and a different detector for each possible signal structure. The higher cost can lead to more accurate detection and a lower false alarm rate. Most spectrum sensing methods assume a signal transmitted over a channel with AWGN and many different types of moving channel models of varying speeds.

3.3.1 Energy Detection

The energy detection based approach for spectrum sensing is one of the simplest methods and is performed by thresholding a spectrogram, that is, any frequency with signal energy higher than the specified threshold is considered to be spectrum in use. Since it does not use any particular features of the waveform, energy detection is a blind method. In the scenario assumed in this thesis, a secondary user could use energy thresholding to detect spectrum in use by a primary user and not use the frequencies that have energy above the threshold. The threshold is based on comparing the expected output signal power to the noise floor. The primary problem with using an energy detector is for choosing a proper threshold. If the threshold is chosen dynamically by measuring the noise floor, then it will not be as effective for spread spectrum signals whose transmissions look like the noise floor.

To understand how the energy detection algorithm functions, an example of time-domain energy detection will first be shown. Assume that a signal, $x(n)$, with sample index, n, has been transmitted through an AWGN channel represented by $w(n)$ and is received as $y(n)$, as given in Equation (25):

$$y(n) = x(n) + w(n) \tag{31}$$

The decision of whether or not the signal is transmitting is based on the signal energy over N received samples:

$$M = \sum_{n=0}^{N} |y(n)|^2, \tag{32}$$

Two hypotheses can be made at any location in the observed signal [6]:

$$\mathcal{H}_0 : \text{Only noise is present.} \tag{33}$$

$$\mathcal{H}_1 : \text{Both signal and noise are present.} \tag{34}$$

The observations $y(n)$ can thus be rewritten as:

$$y(n) = \begin{cases} w(n) & H_0 \\ x(n) + w(n) & H_1, \end{cases} \tag{35}$$

where H_0 is true if there is no signal being transmitted at that frequency and time and

H_1 is true when there is a signal being transmitted. If H_0 is true, then the energy should be less than if H_1 is true. Therefore, for some value η, H_1 can be assumed to be present if $y(n) > \eta$. If η is too small, then the probability of a false positive (P_f), will be high while if η is too high, then the probability of detection P_d, will be small. To modify energy detection for a spectrogram, we begin with the same observations mentioned in Equation 33:

$$S(f,n) = \begin{cases} |W(f,n)|^2 & H_0 \\ |X(f,n) + W(f,n)|^2 & H_1. \end{cases} \tag{36}$$

Similar to the time domain version presented previously, energy thresholding is performed by choosing some η and using the rule that H_1 is assumed to be present if $S(f,n) > \eta$. The same conclusions about the probabilities of false positives and detection can then be made.

3.3.2 Waveform Detection

Since most wireless standards have known features such as preambles, pilot tones, spreading sequences, etc., in their waveforms, it is possible to detect a signal by exploiting these features. A preamble is a known message sent at the beginning of a transmitted signal for the receiver to find and use for synchronization. Pilot tones are symbols (either in frequency or time) that are given higher power relative to symbols used to transmit data. Spreading sequences are pseudo-white noise patterns that are scaled by a symbol used to transmit data. The new "chips," samples of the scaled pattern, are then used for the transmission, which looks similar to white noise due to the spreading process. Since waveform detection requires knowledge of a list of protocols, it is not a blind technique. Instead, multiple waveform detector blocks, each with some sequence, $\phi(n)$, would have to be running in parallel to detect any variety of transmissions as seen in Figure 21. This would usually not be a problem since a primary user will be communicating on known frequency bands allocated to them and using a particular set of protocols. The simplest way to look for most of these features is to correlate them with a known pattern and look for peaks in the resulting correlation. An equivalent and common method for waveform detection is the use of matched filters seen in 21 [29].

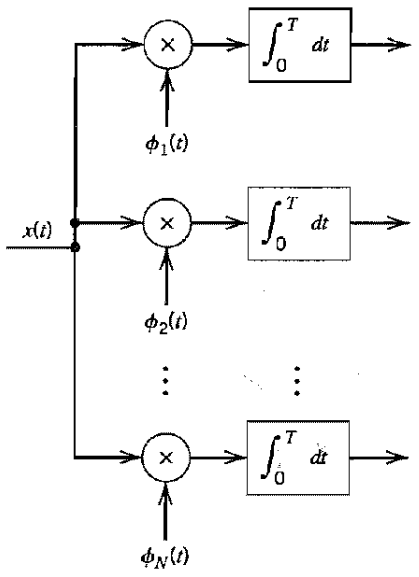


Figure 21: An example of a bank of correlation receivers used for detection [29]. In this diagram $x(t)$ is the input signal and $\phi(t)$ is the signal used for correlation.

To design a correlation receiver, only the desired signal to correlate with is required. Assume a signal transmitted over an AWGN channel as demonstrated in Equation (25). The detection metric is based on the correlation of the received signal with the known signal. First the correlation is performed using some known waveform $h(n)$:

$$M = \left| \sum_{n=1}^{N} y(n)h^*(n) \right| \tag{37}$$

$$= \left| \sum_{n=1}^{N} x(n)h^*(n) + w(n)h^*(n) \right|, \tag{38}$$

where M represents the output of the correlation. In most applications, only the highest value of M that exceeds a threshold is required because that denotes the beginning of the received signal. Once a signal is located, it can be tracked using many methods including correlation with a reference signal and by looking for locations of high energy. When the tracker cannot see the signal anymore, the spectrum is open for use by a cognitive radio.

As mentioned previously, the correlation receiver is equivalent to a matched filter. A matched filter has the time-reversed impulse response of the desired feature used for correlation and takes the form $g(n) = h(N - n)$. So the correlation blocks using $h(n)$ could be replaced with filters using $g(n)$. To prove that a correlation receiver and a matched filter are equivalent start off with the resulting filter output seen in Equation (37):

$$M(n) = \left| \sum_{\nu=1}^{N} y(\nu) h^*(N - n + \nu) \right| . \tag{39}$$

If M is measured at every N samples, then the resulting output will look like:

$$M(N) = \left| \sum_{\nu=1}^{N} y(\nu) h^*(\nu) \right| . \tag{40}$$

Therefore, using a matched filter will be equivalent to using correlations for waveform detection assuming accurate sampling. This method does not have an equivalent method for a spectrogram because all operations are being performed as the signal is incoming. However, this method method could be altered to use FFT-based correlations by taking the FFT of the incoming signal and multiplying it by the frequency domain version of the expected waveforms. Then integrating the result would yield the same result as the matched filter as per Parseval's Theorem, which will be discussed in Section 4.

3.3.3 Cyclostationary Techniques

Since many modern protocols use some form of repetition, looking for cyclostationary features in a transmitted signal is one way of trying to detect a primary user without needing a known waveform. For example, if a preamble or pilot tones are being transmitted regularly, correlating the received signal with a delayed version of itself will show peaks where the pilot tones or preambles are aligning. Additionally, if the protocol uses an OFDM variant like LTE, WiFi, or WiMAX, then there will definitely be some cyclostationary feature because of the cyclic prefix, seen in Figure 22, added before every OFDM symbol is transmitted.

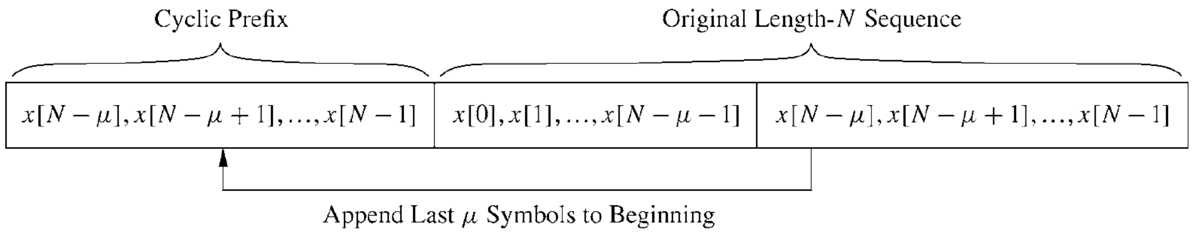

Cyclic Prefix
Original Length-N Sequence
$x[N-\mu], x[N-\mu+1], \ldots, x[N-1]$
$x[0], x[1], \ldots, x[N-\mu-1]$
$x[N-\mu], x[N-\mu+1], \ldots, x[N-1]$
Append Last μ Symbols to Beginning

Figure 22: Example of cyclic prefix of length. μ. being appended to the beginning of an OFDM signal [30].

Instead of calculating a power spectral density (PSD) of the received signal in order to do the spectrum detection, a cyclic spectral density (CSD) is calculated with:

$$S(f,\alpha) = \sum_{\tau=-\infty}^{\infty} R_y^{\alpha}(\tau) e^{-j2\pi f\tau} \tag{41}$$

$$R_y^{\alpha} = E[y(n+\tau)y^*(n-\tau)e^{j2\pi\alpha n}]. \tag{42}$$

The maxima of $S(f,\alpha)$ that exceed a threshold are locations in which a primary user is using spectrum f [6].

4 Denoising Spectrum With Non-local Means

Up till now prior research has only focused on wavelet-based denoising on spectrograms for spectrum sensing. Non-local means (NLM) is the basis for some for many of the most popular image denoising methods and a spectrogram can be thought of as an image. Therefore, NLM should be able to denoise the spectrogram. However, if the added noise in the time domain is AWGN, then the magnitude of the spectrogram is not AWGN. As such, NLM can be modified in order to reflect the new noise models.

One condition that is required before any NLM variant is performed is that the variance of the noise in the frequency domain is known. As mentioned in Section 3, part of the STFT operation is performing a Fourier transform, more specifically the Discrete Fourier Transform (DFT). One of the properties of the DFT is that it obeys Parseval's theorem which states that given a function $x(n)$ with respective DFT X and support N, that is:

$$X_k = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, k \in \mathbb{N}. \tag{43}$$

The squared sums of x and X are related by:

$$\sum_{n=0}^{N-1} |x[n]|^2 = \frac{1}{N} \sum_{k=0}^{N-1} |X[k]|^2. \tag{44}$$

To prove Equation (44), begin by expanding $|X[k]|^2$:

$$\begin{aligned} |X[k]|^2 &= X[k]X^*[k] && (45)\\ &= \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} \sum_{n'=0}^{N-1} x^*[n'] e^{j2\pi n'k/N} && (46)\\ &= \sum_{n=0}^{N-1} x[n] \sum_{n'=0}^{N-1} x^*[n'] e^{j2\pi(n-n')k/N}. && (47) \end{aligned}$$

Now find the energy in the signal over N samples by summing $|X[k]|^2$ over $k \in [0, N-1]$:

$$\sum_{k=0}^{N-1} |X[k]|^2 = \sum_{k=0}^{N-1}\sum_{n=0}^{N-1} x[n] \sum_{n'=0}^{N-1} x^*[n'] e^{j2\pi(n-n')k/N} \tag{48}$$

$$= \sum_{n=0}^{N-1} x[n] \sum_{n'=0}^{N-1} x^*[n'] \sum_{k=0}^{N-1} e^{j2\pi(n-n')k/N}. \tag{49}$$

The innermost sum is a geometric series and can be simplified to:

$$\sum_{k=0}^{N-1} e^{j2\pi(n-n')k/N} = \frac{e^{j2\pi(n-n')} - 1}{e^{j2\pi(n-n')/N} - 1}. \tag{50}$$

Since $k \in \mathbb{Z}$, all values of $(n - n') \in \mathbb{Z}$. Therefore, the the summation in Equation (50) is zero unless $n = n'$:

$$\sum_{k=0}^{N-1} e^{j2\pi(n-n')k/N} = N\delta_{nn'}. \tag{51}$$

It follows that:

$$\sum_{k=0}^{N-1} |X[k]|^2 = N \sum_{n=0}^{N-1} x[n] \sum_{n'=0}^{N-1} x^*[n'] \delta_{nn'} \tag{52}$$

$$= N \sum_{n=0}^{N-1} |x[n]|^2. \tag{53}$$

Notice the scaling by N in front. If the Discrete Fourier transform is defined as:

$$X[k] = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}, \tag{54}$$

$$x[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X[k] e^{j2\pi nk/N}, \tag{55}$$

Then the Fourier transform is unitary and the variance is preserved between transforms. The results of Parseval's Theorem will be used to preserve variance after calculating the spectrogram for NLM. To calculate the variance:

$$
\begin{aligned}
Var(x[n]) &= E[(x[n] - E[x[n]])^2] && (56)\\
&= E[x[n]^2 - 2x[n]E[x[n]] + E[x[n]]^2] && (57)\\
&= E[x[n]^2] - 2E[x[n]]E[x[n]] + E[x[n]]^2 && (58)\\
&= E[x[n]^2] - E[x[n]]^2. && (59)
\end{aligned}
$$

AWGN has the property that the mean is zero, that is $E[x[n]] = 0$:

$$
\begin{aligned}
Var(x[n]) &= E[x[n]^2] && (60)\\
&= \sum_{n=0}^{N-1} |x[n]|^2 && (61)
\end{aligned}
$$

Therefore, by Parseval's theorem, the variance of the noise distribution in the time domain is equal to the variance of the noise distribution in the frequency domain scaled by $\frac{1}{N}$. Because the variance of the noise added in the time domain is assumed known and will not be calculated, the unitary version of the Fourier transform should be used in order to keep the variance the same between the time and frequency domain calculations:

$$
\begin{aligned}
Var(x[n]) &= Var\left(\frac{1}{N}X_k\right) && (62)\\
&= \frac{1}{\sqrt{N}}Var\left(X_k\right). && (63)
\end{aligned}
$$

4.1 Gaussian Noise Model

The complex elements that make up the spectrogram are complex Gaussian random variables. In addition to the noise, there are a variety of factors that could affect each symbol at the receiver such as frequency rotation, the channel model and phase offset, which will cause the real and complex parts to not be independent of each other. Since the real and imaginary parts of the complex valued elements are dependent, the distribution is actually a Bivariate Gaussian. If just a Gaussian channel model is used and it is assumed that frequency rotation and phase offset will not be a problem because there is little time dependence on choosing a start and end time for the spectrogram, then the real and imaginary parts can be assumed to be independent. If that is the case, non-local means can be performed on the real and imaginary parts separately and then added together in order to find the magnitude when performing the energy thresholding. The end result of processing the spectrogram is:

$$Spectrogram = \Re\{NL(spectrogram)\}^2 + \Im\{NL(spectrogram)\}^2. \tag{64}$$

If independence is not assumed, then this method would not work because the influence the real and imaginary parts of the complex number have on each other will be meaningful. The other major flaw in performing NLM on complex numbers is that even though a signal's power is often uniform over the transmitted signal, the symbols can look entirely different. For example, assume quadrature phase shift keying (QPSK), a transmission scheme with symbols $\{-1-j, -1+j, 1-j, 1+j\}$, is used for modulation of the transmitted bit sequence. If that is the case, then it is possible to be in situations where even thought the imaginary and real components have the same transmitted power, the observed symbols would be $1+j$ and $-1-j$, which could hurt finding similarities between different patches due to the different signs on the real and imaginary parts of the symbol. This method will work better for spread-spectrum as opposed to OFDM because the chipping sequence should allow for patches of the same size as the chipping sequence to look similar.

4.2 Rician Noise Model

Since the spectrogram is the magnitude of many complex Gaussian random variables, it fits the description of a Rician-distributed noise model as discussed in Section 2.1.3. Because the noise model changes when the spectrogram is changed, the non-local means algorithm should be changed accordingly to reflect the modification from a Gaussian random variable to the new Rician distribution. The desired value to be estimated is ν since that is proportional to the energy in the real and imaginary components before the magnitude is computed. This step can be performed by modifying the weight estimation used in Equations (14) and (15). These equations assume a maximum likelihood (ML) estimator based on a Gaussian random variable. It is shown in [9] that the estimator can be modified to an ML estimator of a random variable with a different probability distribution.

Because the first moment of a Rician random variable is not easily invertible as it contains a Laguerre polynomial that depends on Bessel functions as seen in Equation (8), a different estimator must be used. The second-order moment is invertible and can be used to create the new estimator as developed in [9]. The second moment of a Rician distribution is:

$$E[x^2] = \int_{-\infty}^{\infty} x^2 p(x|\nu,\sigma)dx \tag{65}$$

$$= \int_{-\infty}^{\infty} x^2 \left(\frac{x}{\sigma^2} e^{-\frac{x^2+\nu^2}{2\sigma^2}} I_0\left(\frac{x\nu}{\sigma^2}\right)\right) dx \tag{66}$$

$$= 2\sigma^2 + \nu^2, \tag{67}$$

where x is the Rician random variable and $p(x|\nu,\sigma)$ is the probability density function of a Rician random variable. Two changes can be made to the NLM algorithm so that ν can be estimated. First, modify Equation (16) to calculate the second moment:

$$NL[v](i) \approx E[x^2] \tag{68}$$

$$NL[v](i) = \sum_{j\in I} w(i,j)v(j)^2. \tag{69}$$

Second, in order to estimate ν, Equations (67) and (69) will be combined:

$$E[x^2] = 2\sigma^2 + \nu^2 \tag{70}$$

$$\nu = \sqrt{E[x^2] - 2\sigma^2} \tag{71}$$

$$\approx \sqrt{\sum_{j \in I} w(i,j) v(j)^2 - 2\sigma^2}. \tag{72}$$

$v(j)$ is squared because the estimator is based on the second moment instead of the first. As mentioned prior, the variance, σ^2, is scaled by $\frac{1}{N}$ due to the NLM variant being performed in the frequency domain. The only problem with the modification presented in this method is that the variance of the Gaussian noise must now be known. Compared to requiring knowledge about the protocol in use and having a detection algorithm for that protocol, requiring knowledge of the variance is not as bad in comparison. It is possible to estimate the variance based on the noise floor or by the fixed point formula for SNR [31]. The latter method could be performed on the patches used for a single iteration of NLM to find a local variance. Experiments described in the Section 5 tests will assume some method for estimating variance is in use and therefore, accurate knowledge of variance will be assumed.

4.3 Non-central χ^2 Noise Model

If the spectrogram is performed without taking the square root such that every element is the magnitude squared of a complex Gaussian random variable, then, with some modification, the noise model of the spectrogram becomes a non-central chi-squared random variable with k degrees of freedom. Since two values, the real and imaginary components of the complex valued elements, are being used to find the spectrogram before NLM is performed, $k = 2$. When the estimation is being performed, the magnitude of the estimated value is taken and modified for the non-central chi-squared distribution in a manner similar to the Rician method presented previously. Unlike a Rician-distributed random variable, an estimate of a non-central chi-squared random variable can be made based on the first moment. The mean of a non-central chi-squared random variable with k degrees of freedom is:

$$E[x] = k + \lambda \tag{73}$$

$$= k + \sum_{i}^{k} \mu_i^2 \tag{74}$$

where λ is the parameter that is estimated as energy in the spectrum since it is proportional to the energy, the variance is considered known. The entire spectrogram is assumed to be corrupted with noise of the same distribution. Therefore $\sigma_i = \sigma$ and so all elements in the spectrogram must be divided by σ^2 in order to complete the data's transformation into a non-central χ^2 random variable. Similar to the Rician method, instead of averaging many Rician random variables and estimating the second moment, the first moment is being evaluated and then adjusted. The derivation of NLM for a noncentral χ^2 random variable:

$$E[x] = \sum_{j \in I} w(i,j) \frac{v(j)}{\sigma^2} = \lambda + k. \tag{75}$$

$$NL[v](i) = \lambda = \sum_{j \in I} w(i,j) \frac{v(j)}{\sigma^2} - 2. \tag{76}$$

The result is multiplied by the variance in order to undo the transformation to noncentral chi-squared and preserve signal power. The assumptions made in this method are that all the complex elements of the spectrogram had similar means and similar powers. As mentioned previously, the spectrogram must be scaled by $\frac{1}{N}$ since this method is being performed in the frequency domain. This method supresses areas that only consist of noise because the real and imaginary parts in those areas will be 0.

One major problem with this method is that frequency offset from the bin frequencies used for the FFT in the spectrogram could apply unwanted rotations in the symbols, which could lead to lower than expected powers. This means that a lower energy threshold may need to be used to guarantee successful energy detection. These problems will be discussed further in the Section 6 and can be resolved empirically with a receiver operating characteristic curve.

5 Simulation

The cognitive radio simulation system has three parts: spectrum generation, denoising, and spectrum detection. The purpose of this section is to discuss how the three parts are combined and implemented. All code used for the simulation is provided in the Appendix.

5.1 Data Generation

Both simulated and live data were generated for the tests. The simulated data was generated in order to look like a random band of spectrum with power. The simplest case was created by first generating one OFDM symbol with the subcarriers of the symbol being considered spectrum in use. An IFFT of the symbol was taken and then many repetitions of the time-domain version of the symbol were concatenated. This basic transmission models one user transmitting with constant power. In order to test denoising against an industry standard, WiFi and Bluetooth data were generated using MATLAB's 802.11 and Bluetooth examples. Once the data was generated, it was corrupted with AWGN before denoising. The WiFi data allowed testing the performance of denoising with spread spectrum when 802.11b was transmitted and denoising with OFDM when 802.11g was transmitted. Bluetooth provided a method of testing the denoising algorithms' performance with frequency hopping. In addition, testing with frequency hopping provides the ability to judge how denoising is affected by a transmitter changing frequency within the same spectrogram.

Only AWGN was applied to the transmitted signals. Other possible channel effects that could have been tested, but were not include the use of a non-AWGN channel and frequency or phase offsets. Frequency and phase offsets are not needed for testing since the energy detection is based on a spectrogram and needing to decode the data is not a concern. A phase offset would not alter the magnitude of the spectrogram. Frequency offset can not be adjusted for without extra knowledge of the signal. Frequency offset would create a spectrogram shifted by a predictable amount in frequency, which would lead to the results being shifted in frequency by the same amount. Applying a stationary or non-stationary channel would just color the signal. Since the signal does not need to be decoded, it should not have any major effects on the spectrogram. The only concern would be nulls in the received signal caused by the channel, but with the binning done by the FFT for the spectrogram, any null should merge with another bin. Alternatively, the seen null would be very narrow and over a short period of time, and that could removed with some extra post-processing.

5.2 Detection Method

Energy detection was used because it is the simplest to implement, has the lowest complexity, and is transmission feature agnostic. In addition, it offered a way to directly compare results with [7], which also used energy detection. However, the method of spectrum detection should not matter as denoising is a preprocessing technique meant for any detection method that does not require the decoding of a signal. In order to study the performance of NLM based denoising of the spectrum using energy detection, a receiver operating characteristic (ROC) curve was calculated for each denoising method and used to determine what would be a good threshold. In order to compare different methods, the area under the curve (AUC) of a ROC was generated. A method was considered better performing if it its AUC approached 100% at a lower SNR.

6 Simulation Results and Evaluation

This section summarizes the results of simulations done over the data described in Section 5. Results are displayed using an area under the curve (AUC) plot and corresponding receiver operating characteristic curves.

6.1 Simulated OFDM Symbols

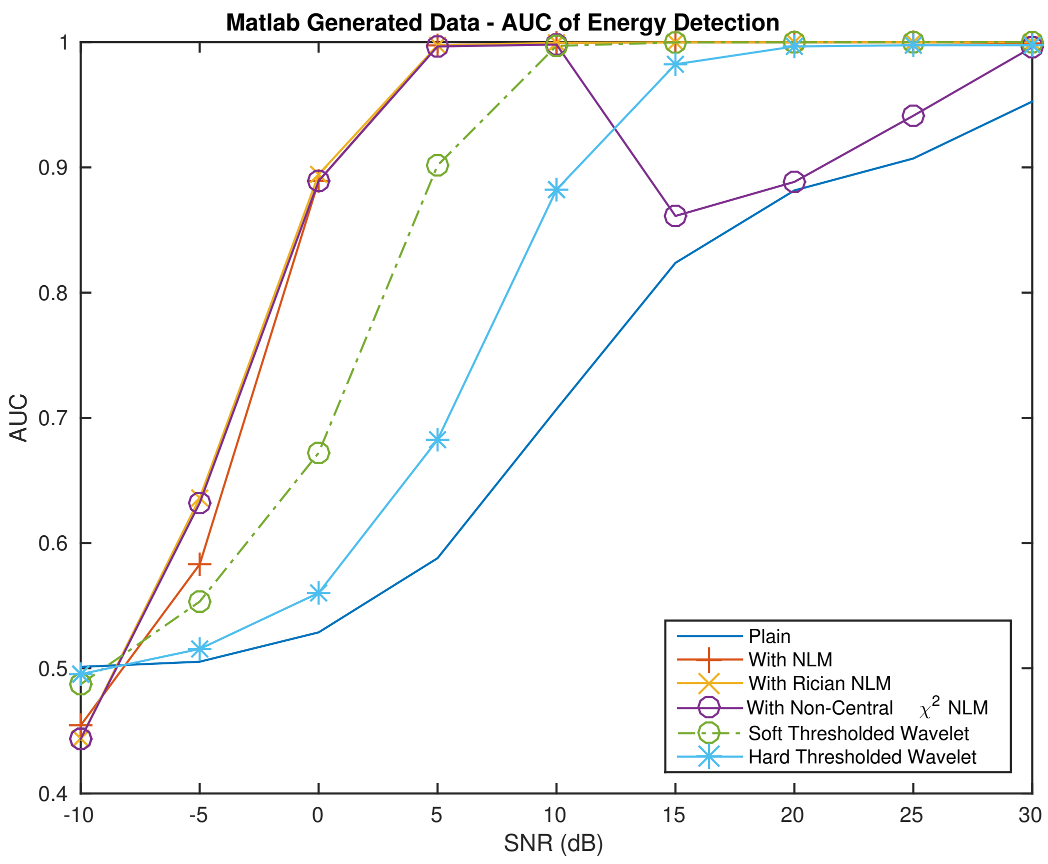


Figure 23: Area under the receiver operating curve for the simulation using OFDM Symbols.

Figure 23 shows the AUC of receiver operating curves ROCs in the simulation for OFDM symbols described in Section 5.1 with energy detection as the spectrum sensing method of choice. AUCs provide a way for visualizing the ROC at a specified SNR. If the ROC output was linear, that is, it looked like the curve $y = x$, then the AUC output would be 0.5. A ROC with this type of curve translates to the algorithm used being no better than blindly guessing. If the AUC is less than 1, then the detection algorithm is needs to reverse all predictions, i.e., a 1 becomes a 0 and 0 becomes 1. Ideally, the AUC is 1 at the lowest SNR possible. This translates to a ROC achieving 1 with a very small false positive rate. No method achieved an AUC of 1. The two closest methods were Rician NLM and plain NLM. After -5dB, these two methods had a higher AUC than all other curves until approximately 10dB SNR when the soft-thresholded wavelet curve approached the NLMR curves. At low SNRs the NLM curves all had an AUC less than .5 meaning that at low SNRs, NLM can be improved by reversing the prediction. In reality, not many signals are going to exist at that high of SNR unless the receiver is close to the transmitter and possibly has line of sight.

One of the more interesting features to note is the similarity between the spectrograms of the received and transmitted signals in Figures 24-26. At -5dB, the transmitted signal is invisible behind all the noise. At 25dB SNR, the features of the transmitted signal are visible in the received signal leading to a significantly better receiver operating curve. The increase in ROC performance can be seen in the AUC graph. At -5dB, the slope of the Rician NLM and MATLAB Wavelet curves increase and quickly beat the performance of everything else. By 0dB, the Rician and MATLAB Wavelet curves already have an AUC of 80% while every other curve is at least 10% lower.

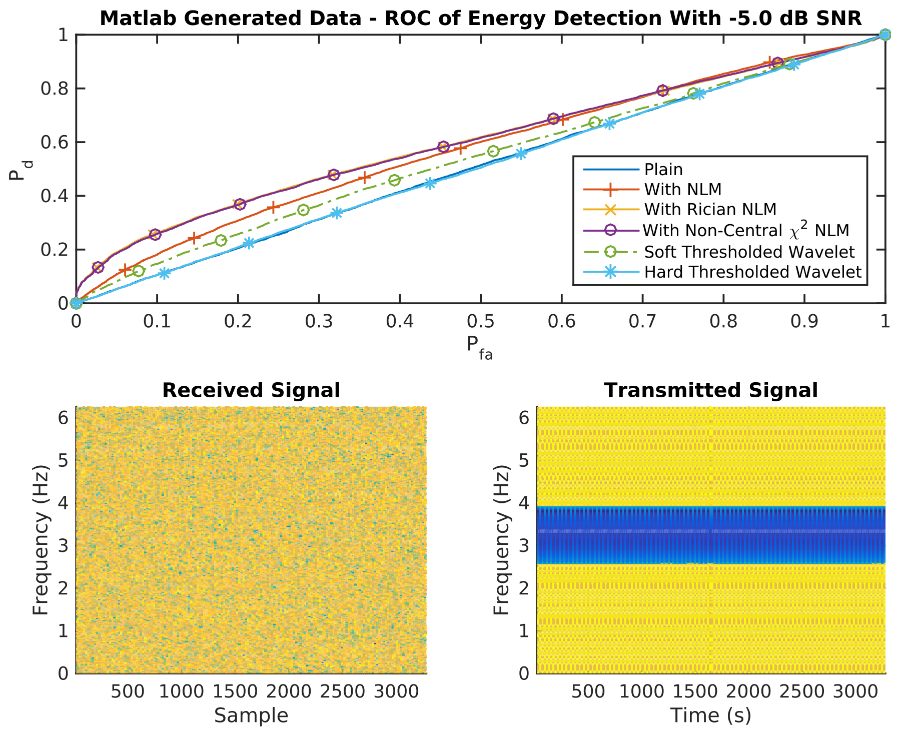


Figure 24: Receiver operating curves at -5dB SNR for the simulation using OFDM symbols.

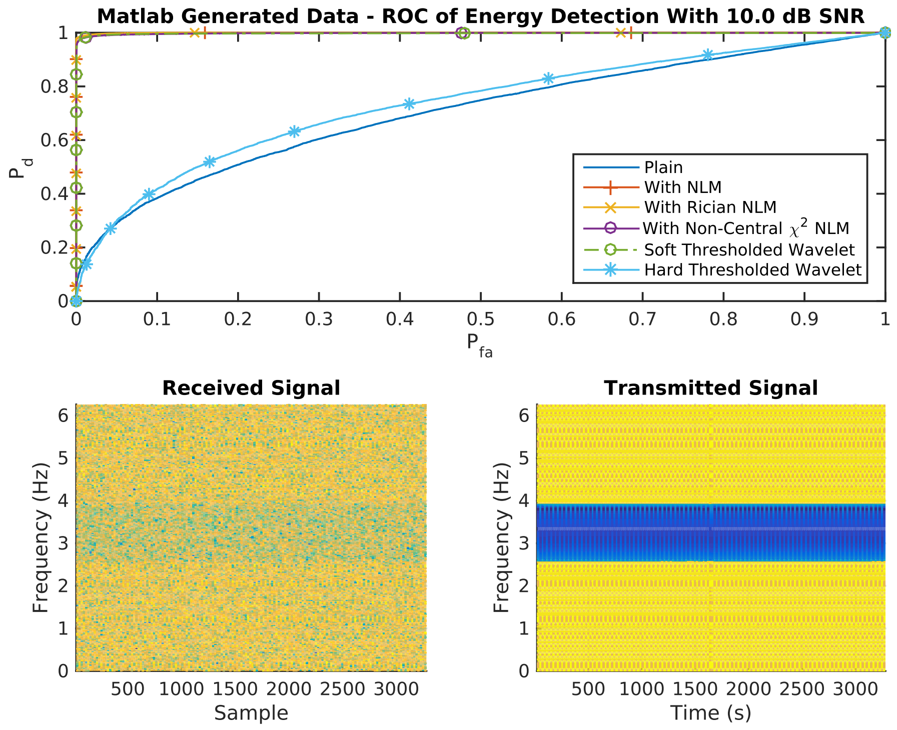


Figure 25: Receiver operating curves at +10dB SNR for the simulation using OFDM symbols.

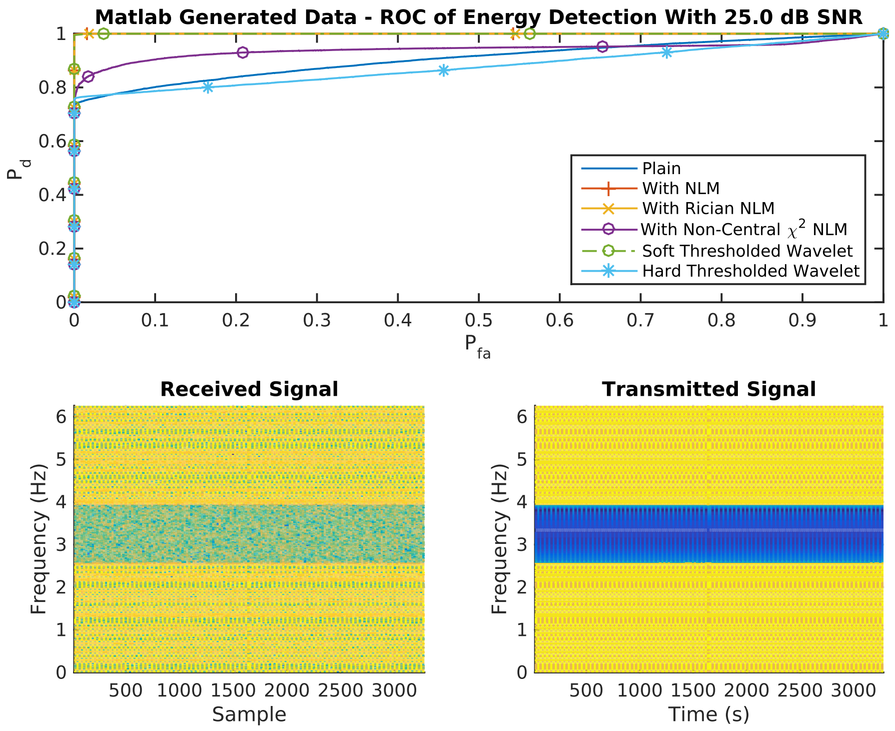


Figure 26: Receiver operating curves at +25dB SNR for the simulation using OFDM symbols.

6.2 Simulated 802.11g Transmission

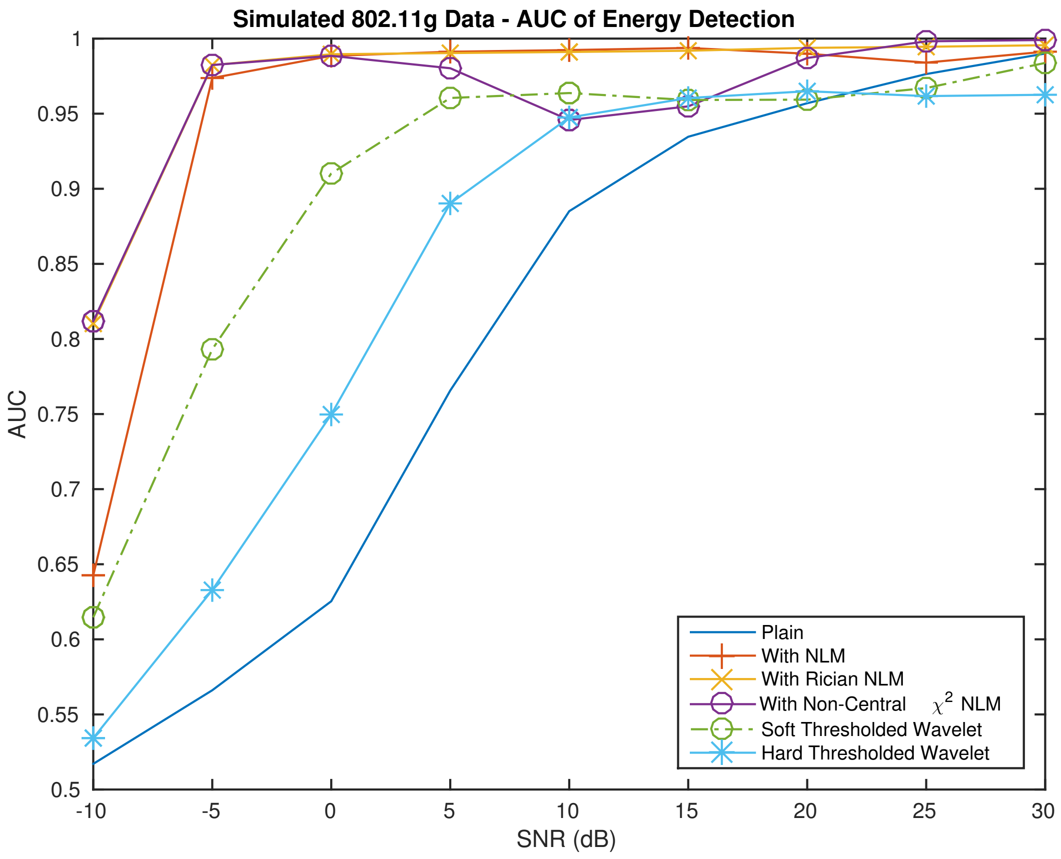


Figure 27: Area under the receiver operating curve for the simulation using MATLAB-generated 802.11g data.

The 802.11g data is the most realistic OFDM signal that was tested. Since the simulated OFDM signals were constant in the frequency domain, they looked more like a spread spectrum signal. The results here show that all NLM variants, particularly the noncentral chi-squared and Rician variants, outperform every other implementation significantly. By -5dB, the AUC of all NLM-based detection methods were over .95 while the soft-thresholded wavelet denoiser was at 0.8. It can also be assumed that the Rician and noncentral chi-squared NLM will have a higher AUC at SNRs lower than -10dB. It is only at 30dB that all algorithms begin to converge. The hard-thresholded wavelet algorithm reached a ceiling of 0.95 showing that hard-thresholding eventually has limits.

The bandwidth of the transmitted signal is much less than that of the signal in the results in Section 6.1.

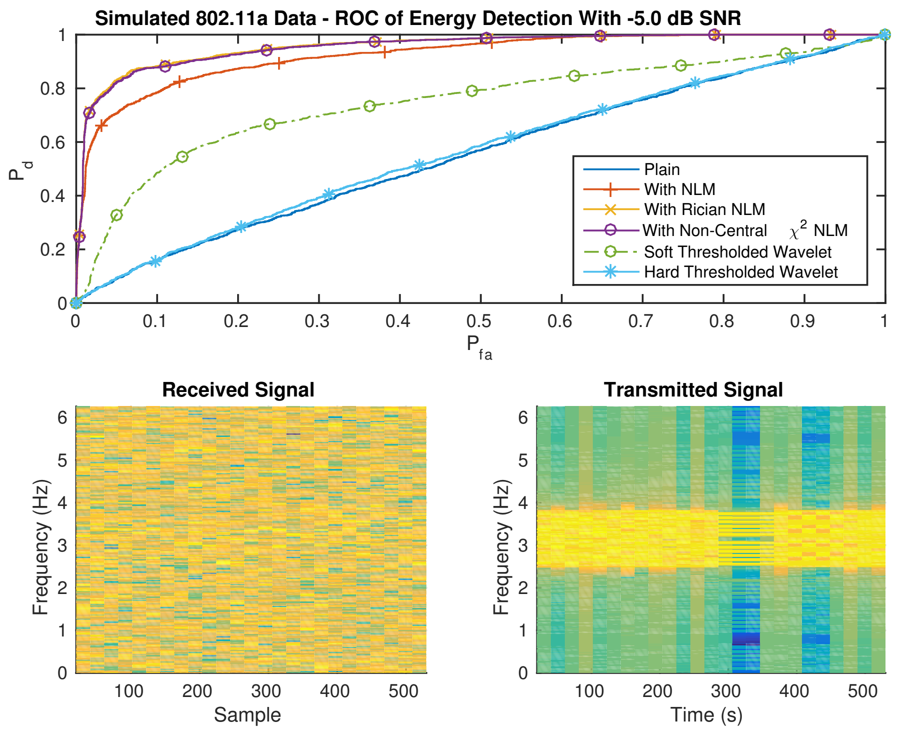


Figure 28: Receiver operating curves at -5dB SNR for the simulation using MATLAB-generated 802.11g data.

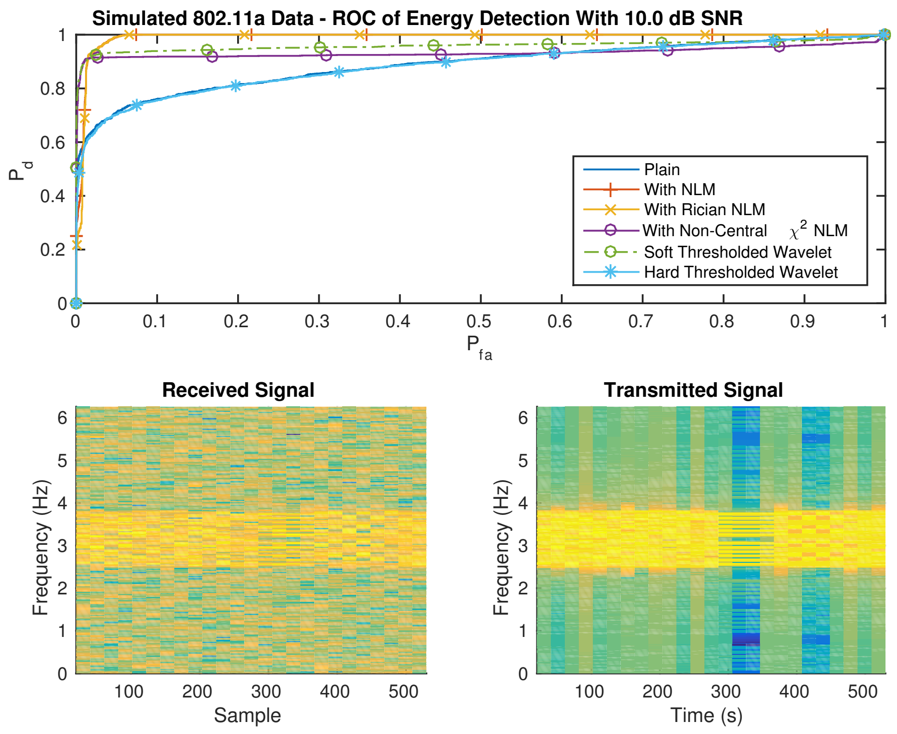


Figure 29: Receiver operating curves at +10dB SNR for the simulation using Matlab-generated 802.11g data.

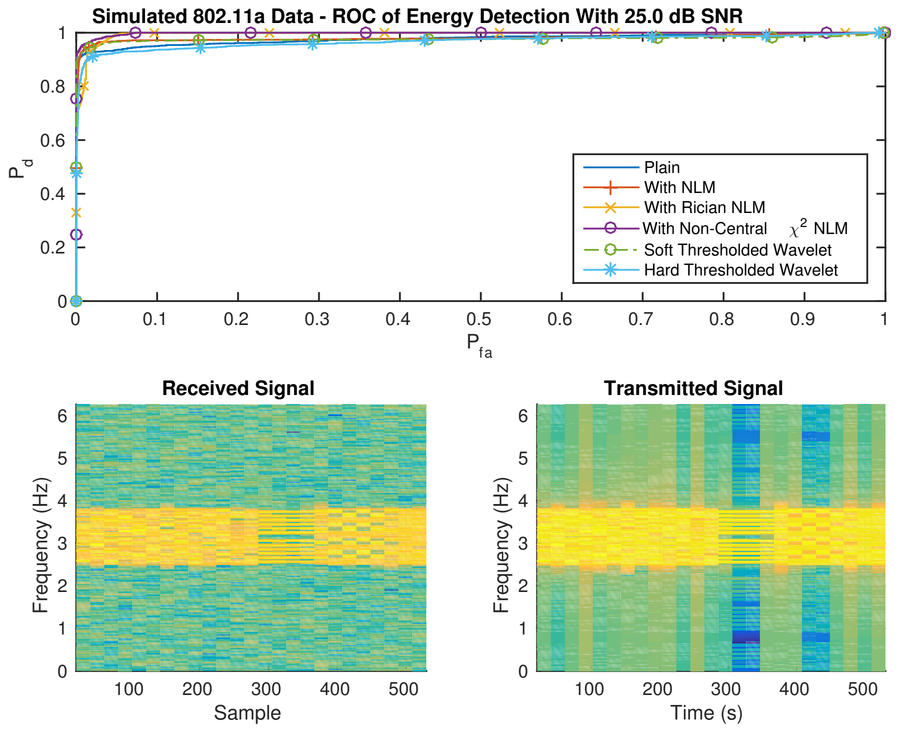


Figure 30: Receiver operating curves at +25dB SNR for the simulation using Matlab-generated 802.11g data.

6.3 Simulated 802.11b Transmission

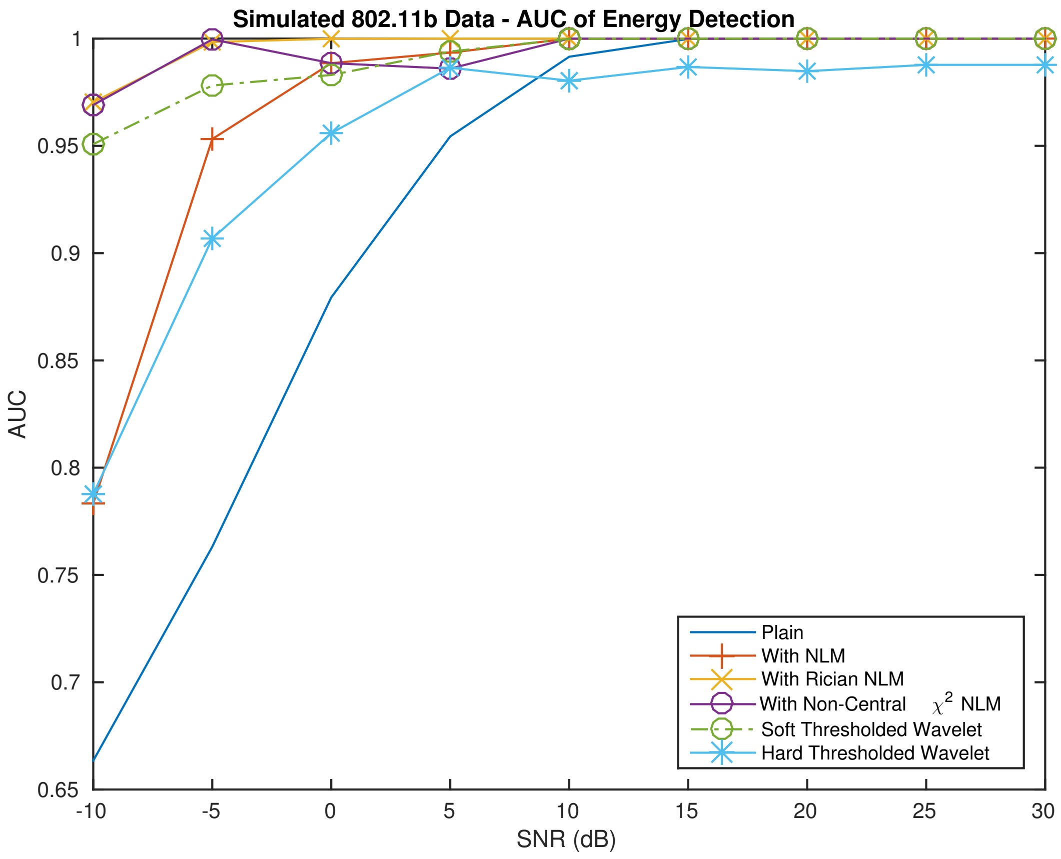


Figure 31: Area under the receiver operating curve for the simulation using MATLAB-generated 802.11b data.

The 801.11b signal is the first spread spectrum signal shown in this section. The transmitted signal was using the 2Mbps 802.11b rate. These signals use a barker spreading code and transmit data using differential quadrature phase shift keying (DQPSK) [18]. Even though the soft-thresholding wavelet algorithm performed better with 802.11g, Rician and noncentral chi-squared again performed better almost universally. All curves converged at around 10dB. Hard-thresholding had a similar problem as with 802.11g, where it reached a ceiling of 0.95 by the time every other curve had an AUC of 1. SNRs less than or equal to 0dB were tested because they are possible to achieve at 802.11b's lowest transmission rate (1Mbps). Less than 0dB is possible if the cognitive radio in use is far away from the primary user's radio and both are in a bad environment or moving. The unmodified NLM algorithm did not perform quite as well as the NLM variants and converged with soft-thresholded wavelet denoising at 0dB.

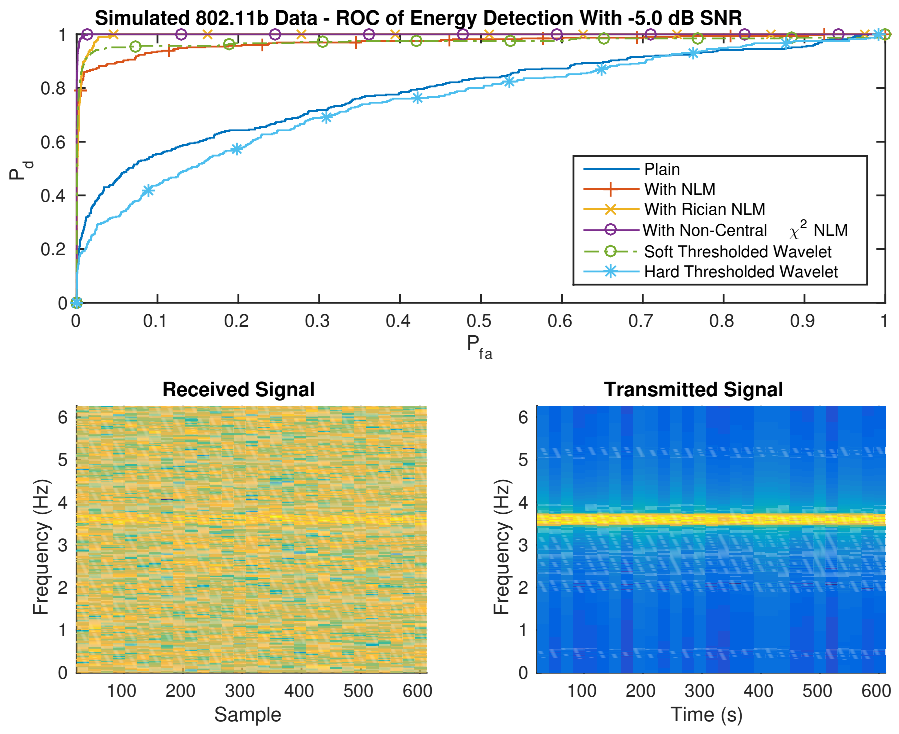


Figure 32: Receiver operating curves at -5dB SNR for the simulation using Matlab-generated 802.11b data.

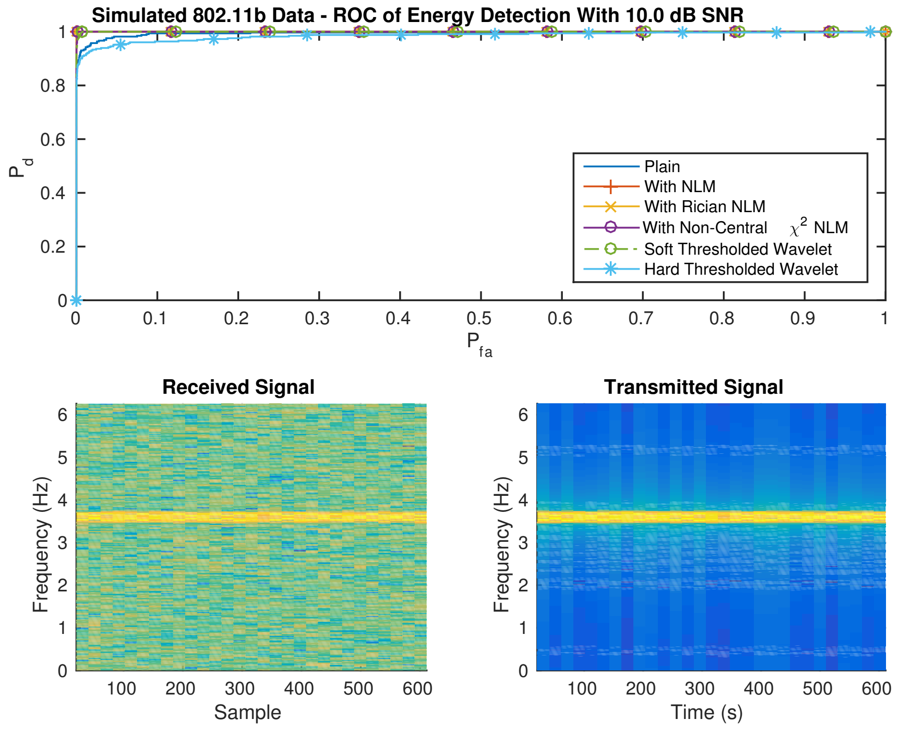


Figure 33: Receiver operating curves at +10dB SNR for the simulation using Matlab-generated 802.11b data.

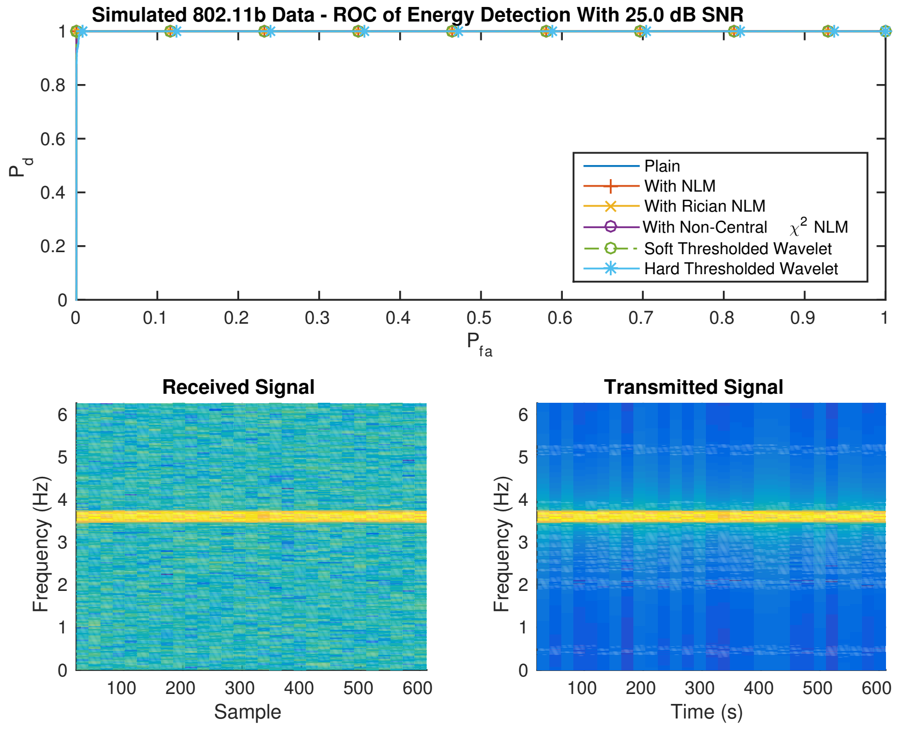


Figure 34: Receiver operating curves at +25dB SNR for the simulation using Matlab-generated 802.11b data.

6.4 Simulated Bluetooth Transmission

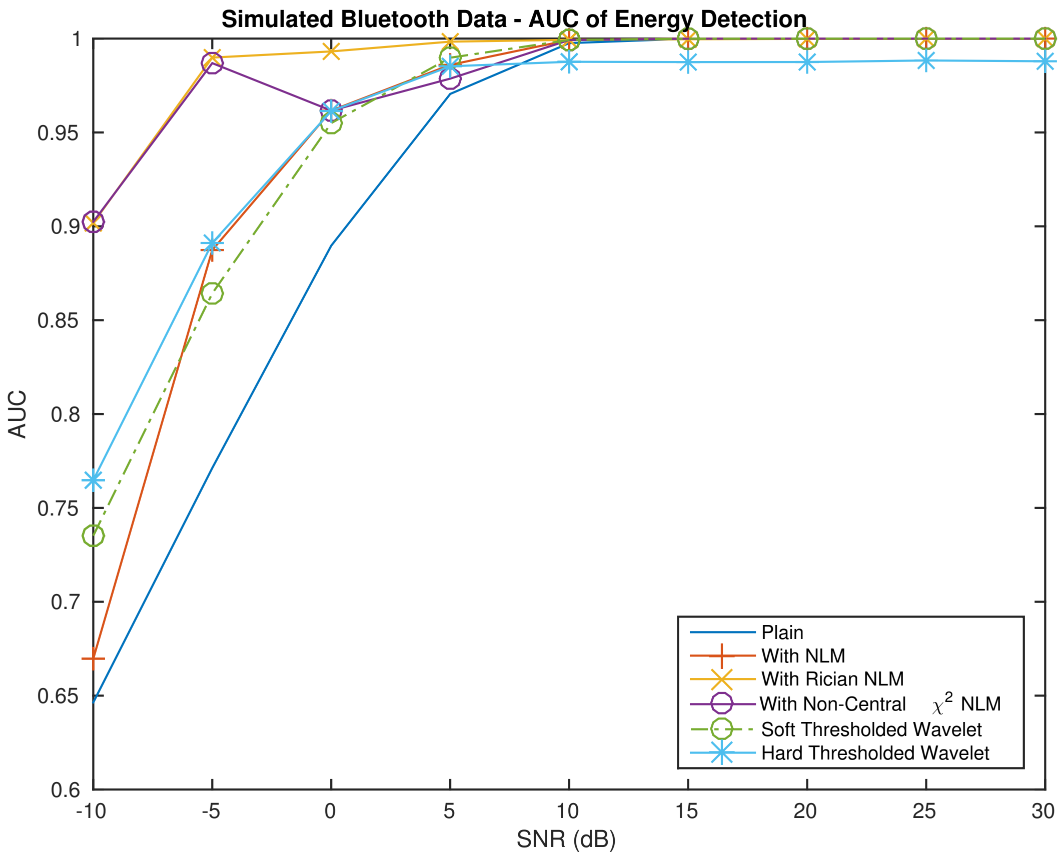


Figure 35: Area under the receiver operating curve for the simulation using MATLAB-generated Bluetooth data.

All algorithms had good performance when denoising Bluetooth as seen in Figure 35. As in all previous tests, NLM achieved a probability of detection of 1 with the lowest probability of false alarm in comparison to every other method. Rician NLM had better performance overall compared to noncentral chi-squared. Again hard-thresholded wavelets obtained a ceiling of .95 AUC. By 0dB, all denoising methods except for Rician NLM converged (Rician NLM was doing better at that point). In addition, all curves maintained an AUC greater than .5 for the entire SNR range. The unmodified NLM algorithm performed worse in this test compared to the other algorithms as it converged with the wavelet methods at -5dB, 5dB less than the convergence point in previous simulations. The improvement in overall performance can be explained by more of the Bluetooth signal present in the -5dB SNR spectrogram seen in Figure 36 than in the higher SNRs seen in Figures 37 and 38.

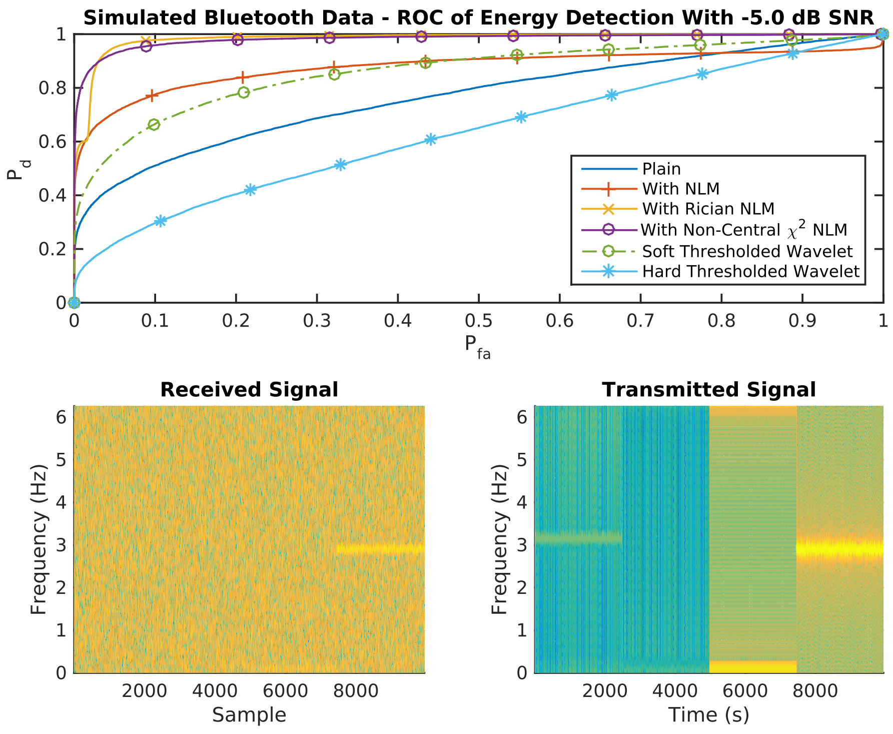


Figure 36: Receiver operating curves at -5dB SNR for the simulation using Matlab-generated Bluetooth data.

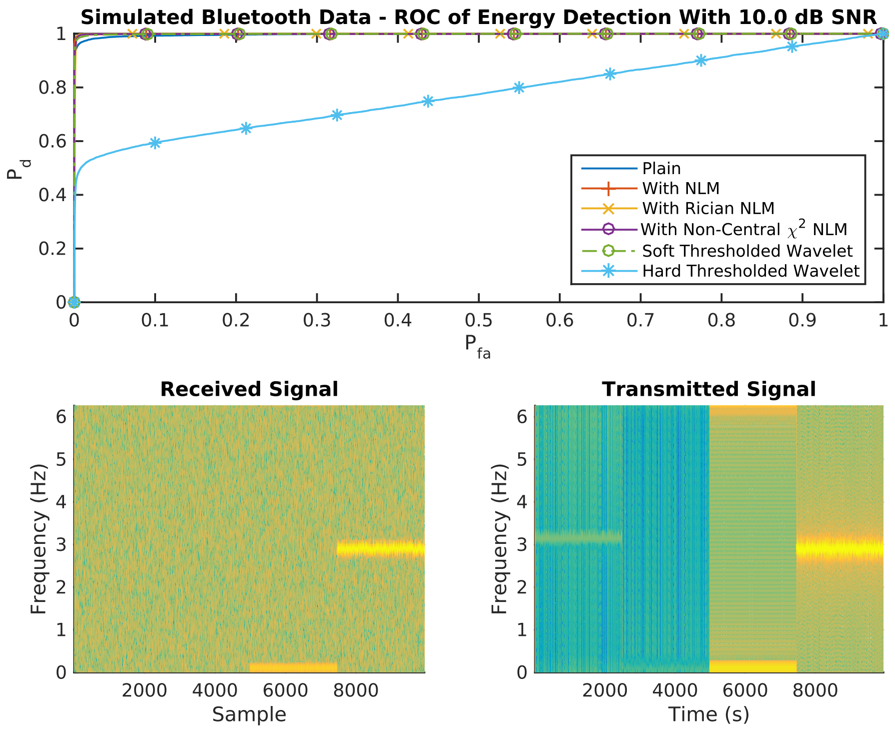


Figure 37: Receiver operating curves at +10dB SNR for the simulation using MATLAB-generated Bluetooth data.

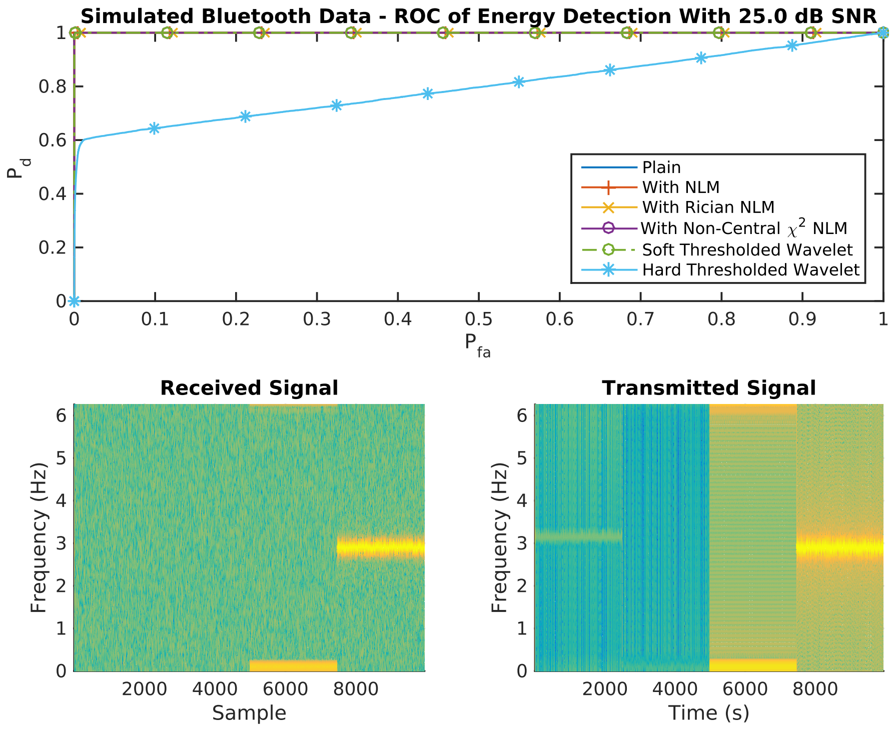


Figure 38: Receiver operating curves at +25dB SNR for the simulation using MATLAB-generated Bluetooth data.

7 Conclusion and Future Work

Even though cognitive radio and smart radio technology are currently not being used with sharing spectrum, the technology looks to be very promising in the future where secondary users must cooperate with primary users and vice versa. Although most methods look at processing raw data for detection of spectrum usage, many do not try to enhance the spectrum for more accurate detection. The only prior works in this area currently use wavelet-based denoising in order to improve the spectrogram. Using the more popular non-local means method for denoising provided better results overall and by tailoring non-local means to the probability distributions of the noise in the spectrograms a new way of preprocessing spectrograms for spectrum usage was developed.

Simulations using various types of waveforms showed that the NLM variants provided better overall performance, even at very low SNRs. These simulations should also be performed with a cognitive radio such as the USRPs in order to confirm the simulations' results. It also showed that using soft-thresholded wavelet denoising, which has not been tried before for this application performed better than hard-thresholding for this application. However, the computational complexity of NLM is significantly worse than that of wavelet denoising as experienced by the longer runtime during simulations. It may also be possible to speed up nonlocal means with the use of GPUs and other parallel computing tools. The runtime of nonlocal means versus other algorithms should be run and quantified. Nonetheless, the connection developed in this thesis between spectrum sensing and NLM provides a new motivation for utilizing other methods developed in image processing for communications and cognitive radio applications.

References

[1] N. Telecommunications and I. Administration, *United States Frequency Allocation.* 2011.

[2] Verizon Wireless, “map-why-vzw-verizon,” 2015. [Online; accessed June 1, 2015].

[3] F. C. Commission, “Fcc proposes rules to facilitate wireless broadband services using vacant tv channels,” May 2004.

[4] Federal Communication Commission, “Fcc 14-144,” 2014.

[5] Institute of Electrical and Electronics Engineers, “802.22-2011,” 2011.

[6] T. Yucek and H. Arslan, “A survey of spectrum sensing algorithms for cognitive radio applications,” *Communications Surveys Tutorials, IEEE*, vol. 11, pp. 116–130, First 2009.

[7] H. Wang, Y. Xu, X. Su, and J. Wang, “Cooperative spectrum sensing with wavelet denoising in cognitive radio,” in *Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st*, pp. 1–5, May 2010.

[8] A. Buades, B. Coll, and J.-M. Morel, “A non-local algorithm for image denoising,” in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05) - Volume 2 - Volume 02*, CVPR ’05, (Washington, DC, USA), pp. 60–65, IEEE Computer Society, 2005.

[9] N. Wiest-Daesslé, S. Prima, P. Coupé, S. Morrissey, and C. Barillot, “Rician noise removal by non-local means filtering for low signal-to-noise ratio mri: Applications to dt-mri,” in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2008* (D. Metaxas, L. Axel, G. Fichtinger, and G. Székely, eds.), vol. 5242 of *Lecture Notes in Computer Science*, pp. 171–179, Springer Berlin Heidelberg, 2008.

[10] A. Papoulis, *Probability, Random Variables and Stochastic Processes.* McGraw-Hill Companies, 3rd ed., Feb. 1991.

[11] S. Aja-Fernandez, C. Alberola-Lopez, and C.-F. Westin, “Noise and signal estimation in magnitude mri and rician distributed images: A lmmse approach,” *Image Processing, IEEE Transactions on*, vol. 17, pp. 1383–1398, Aug 2008.

[12] R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision.* New York, NY, USA: McGraw-Hill, Inc., 1995.

[13] E. Arias-Castro and D. L. Donoho, “Does median filtering truly preserve edges better than linear filtering?,” *The Annals of Statistics*, vol. 37, pp. 1172–1206, 06 2009.

[14] D. Donoho, “De-noising by soft-thresholding,” *Information Theory, IEEE Transactions on*, vol. 41, pp. 613–627, May 1995.

[15] D. L. Donoho and I. M. Johnstone, “Ideal spatial adaptation by wavelet shrinkage,” *Biometrika*, vol. 81, pp. 425–455, 1994.

[16] M. Lavielle, “Detection of multiple changes in a sequence of dependent variables,” *Stochastic Processes and their Applications*, vol. 83, no. 1, pp. 79 – 102, 1999.

[17] Institute of Electrical and Electronics Engineers, “802.15.1-2005,” 2005.

[18] Institute of Electrical and Electronics Engineers, “802.11-2012,” 2012.

[19] 3GPP, “Lte; evolved universal terrestrial radio access (e-utra); physical channel and modulation (3gpp ts 36.211 version 12.6.0 release 12),” 2015.

[20] 3GPP2, “Physical layer standard for cdma2000 spread spectrum systems release a,” 2002.

[21] Ettus Research, 2015.

[22] Ettus Research, “Sdr for beginners: Building an fm receiver with the usrp and gnu radio,” 2015.

[23] GNURadio, “Global positioning system (gps),” 2014.

[24] Open RFID Lab, “Rfid prototyping and testing platform,” 2013.

[25] Ettus Research, 2015.

[26] E. Research, “Usrp hardware driver and usrp manual,” Sept. 2014.

[27] Ettus Research, “Usrp n210,” 2015. [Online; accessed June 28, 2015].

[28] Ettus Research, “Usrp x210,” 2015. [Online; accessed June 28, 2015].

[29] S. Haykin, *Communication Systems.* Wiley Publishing, 4th ed., 2000.

[30] A. Goldsmith, *Wireless Communications.* Cambridge University Press, 1st ed., 2005.

[31] C. G. Koay and P. J. Basser, “Analytically exact correction scheme for signal extraction from noisy magnitude {MR} signals,” *Journal of Magnetic Resonance*, vol. 179, no. 2, pp. 317 – 322, 2006.

Appendix Matlab Simulation Code

A AUC Simulation

```
function result = runSimAuc( sigTx, sinrs, numIter,figTitle,
   saveDir,savePrefix,resampleRate,threshold )
%runSim runs a simulation and returns a struct filled with
   various
%names and denoised spectrograms
%   sigTx - clean signal
%   sinrs - sinrs to test at
%   numIter - number of iterations
t = 15;
%%
%
% * ITEM1
% * ITEM2
%
f = 10;

addpath('../denoisers')
noiseVar = 10.^(-sinrs./10);
hnl = sqrt(noiseVar); % noise variance is known. So that's
   helpful
%resampleRate = 10;
%signalLen = length(sigTx);
%hnl = ones(size(hnl));


sinrsLen = length(sinrs);
auc = zeros(sinrsLen,1);
aucNlm =  zeros(sinrsLen,1);
aucNlmRician =  zeros(sinrsLen,1);
aucNlmNonCenChi2 =  zeros(sinrsLen,1);
aucDwt =  zeros(sinrsLen,1);
aucDwtPaper =  zeros(sinrsLen,1);

mseAuc = zeros(sinrsLen,1);
mseAucNlm =  zeros(sinrsLen,1);
mseAucNlmRician =  zeros(sinrsLen,1);
mseAucNlmNonCenChi2 =  zeros(sinrsLen,1);
mseAucDwt =  zeros(sinrsLen,1);
```

```matlab
mseAucDwtPaper =  zeros(sinrsLen,1);

gammaAuc = zeros(sinrsLen,1);
gammaAucNlm =  zeros(sinrsLen,1);
gammaAucNlmRician =  zeros(sinrsLen,1);
gammaAucNlmNonCenChi2 =  zeros(sinrsLen,1);
gammaAucDwt =  zeros(sinrsLen,1);
gammaAucDwtPaper =  zeros(sinrsLen,1);

mseGammaAuc = zeros(sinrsLen,1);
mseGammaAucNlm =  zeros(sinrsLen,1);
mseGammaAucNlmRician =  zeros(sinrsLen,1);
mseGammaAucNlmNonCenChi2 =  zeros(sinrsLen,1);
mseGammaAucDwt =  zeros(sinrsLen,1);
mseGammaAucDwtPaper =  zeros(sinrsLen,1);

[sigTxMag,F,T,P] = spectrogram(sigTx,64*resampleRate);
sigTxMag = abs(sigTxMag).^2;
sigTxMag = fftshift(sigTxMag);
P = fftshift(P);
sigTxLogical = sigTxMag > threshold;

%normImg = @(img) img./max(max(img));
normImg = @(img) img;
for ii = 1:sinrsLen
    (ii-1)/sinrsLen
    for iter = 0:numIter-1
        %waitbar( ((ii-1)*numIter+iter)/(length(sinrs)*
           numIter))

        %sigRx = awgn(filter(isiChan,wifiaTx),sinrs(ii));
        %sigRx = awgn(wifiaTx,sinrs(ii),'measured');
        sigRx = awgn(sigTx,sinrs(ii));

        sigRx = spectrogram(sigRx,64*resampleRate)./sqrt(64*
           resampleRate);


        sigRx = fftshift(sigRx);

        %in the future consider doing frequency shifts and
           such

        %abs()^2 makes it noncentral chi^2
        sigMag = abs(sigRx).^2;

        %using wavelet denoising
        sigMagDwt = wavelet_denoise_builtin(sigMag,'sym4',3)
            ;

        sigMagDwtPaper = wavDenoisePaper(sigMag,'sym4',2);

        %taking the sqrt xfms it from noncentral chi^2 to
            rician
        %square to convert back
        %sigMagNlm = FastNonLocalMeans3D(sigMag,noiseVar(ii)
            ,hnl(ii),[t,t,1],[f,f,1],'gaussian');
        %sigMagNlmRician = FastNonLocalMeans3D(sigMag,1/
            noiseVar(ii),hnl(ii),[t,t,1],[f,f,1],'rician');

        sigMagNlm = NLmeansfilter((sigMag),t,f,hnl(ii));
        sigMagNlmRician = NLmeansfilterRician(sqrt(sigMag),t
            ,f,hnl(ii),noiseVar(ii)).^2;

        %normalize by the noise variance so it becomes a
            noncentral chi^2 distributed r.v.
        %the magnitude^2 of a complex number with awgn is
            chi^2 with 2 degrees of freedom
        sigMagNlmNonCenChi2 = NLmeansfilterNonCenChi2(sigMag
            ./noiseVar(ii),t,f,hnl(ii),2).*noiseVar(ii);

        %{
        sigMagNlm = FAST_NLM_II((sigMag),t,f,hnl(ii));
        sigMagNlmRician = abs((FAST_NLM_IIRician(sqrt(sigMag
            ),t,f,hnl(ii)))).^2;
        %normalize by the noise variance so it becomes a
            noncentral chi^2 distributed r.v.
        %the magnitude^2 of a complex number with awgn is
            chi^2 with 2 degrees of freedom
        sigMagNlmNonCenChi2 = FAST_NLM_IINonCenChi2(sigMag./
            noiseVar(ii),t,f,hnl(ii),2);
        %switching for loops will probably make it more
            efficient, but less readable
        %}

        %sigTxLogical = logical(sigTxMag);

        auc(ii) = prtScoreAuc(sigMag(:), sigTxLogical(:));
        aucNlm(ii) = prtScoreAuc(sigMagNlm(:), sigTxLogical
            (:));
        aucNlmRician(ii) = prtScoreAuc(sigMagNlmRician(:),
           sigTxLogical(:));
        aucNlmNonCenChi2(ii) = prtScoreAuc(
           sigMagNlmNonCenChi2(:), sigTxLogical(:));
        aucDwt(ii) = prtScoreAuc(sigMagDwt(:), sigTxLogical
           (:));
        aucDwtPaper(ii) = prtScoreAuc(sigMagDwtPaper(:),
           sigTxLogical(:));

        mseAuc(ii) = psnr(normImg(sigMag), normImg(sigTxMag)
           );
        mseAucNlm(ii) = psnr(normImg(sigMagNlm), normImg(
           sigTxMag));
        mseAucNlmRician(ii) = psnr(normImg(sigMagNlmRician),
            normImg(sigTxMag));
        mseAucNlmNonCenChi2(ii) = psnr(normImg(
           sigMagNlmNonCenChi2), normImg(sigTxMag));
        mseAucDwt(ii) = psnr(normImg(sigMagDwt), normImg(
           sigTxMag));
        mseAucDwtPaper(ii) = psnr(normImg(sigMagDwtPaper),
           normImg(sigTxMag));
        %{
        posclass = logical(1);
        [~,~,~,auc(ii)] = perfcurve(sigTxLogical(:),sigMag
           (:),posclass);
        [~,~,~,aucNlm(ii)] = perfcurve(sigTxLogical(:),
           sigMagNlm(:), posclass);
        [~,~,~,aucNlmRician(ii)] = perfcurve(sigTxLogical(:)
           ,sigMagNlmRician(:), posclass);
        [~,~,~,aucNlmNonCenChi2(ii)] = perfcurve(
           sigTxLogical(:),sigMagNlmNonCenChi2(:),posclass);
        [~,~,~,aucDwt(ii)] = perfcurve(sigTxLogical(:),
           sigMagDwt(:),posclass);
        [~,~,~,aucDwtPaper(ii)] = perfcurve(sigTxLogical(:),
           sigMagDwtPaper(:),posclass);
        %}
        if iter==0
            gammaAuc(ii) = auc(ii);
            gammaAucNlm(ii) = aucNlm(ii);
            gammaAucNlmRician(ii) = aucNlmRician(ii);
            gammaAucNlmNonCenChi2(ii) = aucNlmNonCenChi2(ii)
               ;
            gammaAucDwt(ii) = aucDwt(ii);
            gammaAucDwtPaper(ii) = aucDwtPaper(ii);
        else
```

```

            gammaAuc(ii) = gammaAuc(ii) + auc(ii);
            gammaAucNlm(ii) = gammaAucNlm(ii) + aucNlm(ii);
            gammaAucNlmRician(ii) = gammaAucNlmRician(ii) + aucNlmRician(ii);
            gammaAucNlmNonCenChi2(ii) = gammaAucNlmNonCenChi2(ii) + aucNlmNonCenChi2(ii);
            gammaAucDwt(ii) = gammaAucDwt(igammai) + aucDwt(ii);
            gammaAucDwtPaper(ii) = gammaAucDwtPaper(ii) + aucDwtPaper(ii);
        end

        if iter==0
            mseGammaAuc(ii) = mseAuc(ii);
            mseGammaAucNlm(ii) = mseAucNlm(ii);
            mseGammaAucNlmRician(ii) = mseAucNlmRician(ii);
            mseGammaAucNlmNonCenChi2(ii) = mseAucNlmNonCenChi2(ii);
            mseGammaAucDwt(ii) = mseAucDwt(ii);
            mseGammaAucDwtPaper(ii) = mseAucDwtPaper(ii);
        else

            mseGammaAuc(ii) = mseGammaAuc(ii) + mseAuc(ii);
            mseGammaAucNlm(ii) = mseGammaAucNlm(ii) + mseAucNlm(ii);
            mseGammaAucNlmRician(ii) = mseGammaAucNlmRician(ii) + mseAucNlmRician(ii);
            mseGammaAucNlmNonCenChi2(ii) = mseGammaAucNlmNonCenChi2(ii) + mseAucNlmNonCenChi2(ii);
            mseGammaAucDwt(ii) = mseGammaAucDwt(ii) + mseAucDwt(ii);
            mseGammaAucDwtPaper(ii) = mseGammaAucDwtPaper(ii) + mseAucDwtPaper(ii);
        end
    end
end


gammaAuc = gammaAuc ./numIter;
gammaAucNlm = gammaAucNlm ./numIter;
gammaAucNlmRician = gammaAucNlmRician ./numIter;
gammaAucNlmNonCenChi2 = gammaAucNlmNonCenChi2 ./numIter;
```

```
gammaAucDwt = gammaAucDwt ./numIter;
gammaAucDwtPaper = gammaAucDwtPaper ./numIter;

mseGammaAuc = mseGammaAuc ./numIter;
mseGammaAucNlm = mseGammaAucNlm ./numIter;
mseGammaAucNlmRician = mseGammaAucNlmRician ./numIter;
mseGammaAucNlmNonCenChi2 = mseGammaAucNlmNonCenChi2 ./numIter;
mseGammaAucDwt = mseGammaAucDwt ./numIter;
mseGammaAucDwtPaper = mseGammaAucDwtPaper ./numIter;

cmap = lines;
%figure
plot(sinrs,gammaAuc,'Color',cmap(1,:))
hold on
line_fewer_markers(sinrs,gammaAucNlm,10,'-+','Color',cmap(2,:))
line_fewer_markers(sinrs,gammaAucNlmRician,10,'-x','Color',cmap(3,:))
line_fewer_markers(sinrs,gammaAucNlmNonCenChi2,10,'-o','Color',cmap(4,:))
line_fewer_markers(sinrs,gammaAucDwt,10,'-.g','Color',cmap(5,:))
line_fewer_markers(sinrs,gammaAucDwtPaper,10,'-*','Color',cmap(6,:))
legend('Plain','With NLM','With Rician NLM','With Non-Central \chi^2 NLM','Soft Thresholded Wavelet','Hard Thresholded Wavelet','Location','southeast')
%legend('Plain','With NLM','With Rician NLM','With Non-Central \chi^2 NLM','Wavelet Paper')

title(sprintf('%s - AUC of Energy Detection',figTitle));
xlabel('SNR (dB)')
ylabel('AUC')
savefig(sprintf('%s/%s_auc.fig',saveDir,savePrefix))
saveas(gcf, sprintf('%s_eps/%s_auc.eps',saveDir,savePrefix), 'eps2c')

figure
plot(sinrs,mseGammaAuc,'Color',cmap(1,:))
hold on
line_fewer_markers(sinrs,mseGammaAucNlm,10,'-+','Color',cmap(2,:))
line_fewer_markers(sinrs,mseGammaAucNlmRician,10,'-x','Color',cmap(3,:))
line_fewer_markers(sinrs,mseGammaAucNlmNonCenChi2,10,'-o','
   Color',cmap(4,:))
line_fewer_markers(sinrs,mseGammaAucDwt,10,'-.g','Color',
   cmap(5,:))
line_fewer_markers(sinrs,mseGammaAucDwtPaper,10,'-*','Color'
   ,cmap(6,:))
legend('Plain','With NLM','With Rician NLM','With Non-
   Central \chi^2 NLM','Soft Thresholded Wavelet','Hard
   Thresholded Wavelet','Location','southeast')
%legend('Plain','With NLM','With Rician NLM','With Non-
   Central \chi^2 NLM','Wavelet Paper')

title(sprintf('%s - PSNR of Denoised Transmission',figTitle)
   );
xlabel('Received SNR (dB)')
ylabel('Denoised PSNR (dB)')
savefig(sprintf('%s/%s_psnr.fig',saveDir,savePrefix))
saveas(gcf, sprintf('%s_eps/%s_psnr.eps',saveDir,savePrefix)
   , 'eps2c')

figure
sf = surf(T,F,10*log10(P));
sf.EdgeColor = 'none';
axis tight
view(0,90)
title('Transmitted Signal')
xlabel('Sample')
ylabel('Frequency (Hz)')

savefig(sprintf('%s/%s_auc_tx_spec.fig',saveDir,savePrefix))
saveas(gcf,sprintf('%s_eps/%s_auc_tx_spec.eps',saveDir,
   savePrefix),'eps2c')

result=1;
end
```

B ROC Simulation

```
function result = runSimRoc( sigTx, sinr, numIter,figTitle,
   saveDir,savePrefix,resampleRate,threshold )
%runSim runs a simulation and returns a struct filled with
   various
%names and denoised spectrograms
```

```
%   sigTx - clean signal
%   sinrs - sinrs to test at
%   numIter - number of iterations
t = 15;
f = 10;

addpath('../denoisers')
noiseVar = 10.^(-sinr./10);
hnl = sqrt(noiseVar); % noise variance is known. So that's
   helpful
%resampleRate = 10;
%signalLen = length(sigTx);

[sigTxMag,F,T,P] = spectrogram(sigTx,64*resampleRate);
sigTxMag = abs(sigTxMag).^2;
sigTxMag = fftshift(sigTxMag);
P = fftshift(P);
sigTxLogical = sigTxMag > threshold;


for iter = 0:numIter-1

    %sigRx = awgn(filter(isiChan,wifiaTx),sinrs);
    %sigRx = awgn(wifiaTx,sinrs,'measured');
    sigRx = awgn(sigTx,sinr);

    sigRx = spectrogram(sigRx,64*resampleRate)./sqrt(64*
       resampleRate);

    sigRx = fftshift(sigRx);
    %in the future consider doing frequency shifts and such

    %abs()^2 makes it noncentral chi^2
    sigMag = abs(sigRx).^2;

    %using wavelet denoising
    sigMagDwt = abs(wavelet_denoise_builtin(sigMag,'sym4',3)
       );

    sigMagDwtPaper = abs(wavDenoisePaper(sigRx,'sym4',2));

    %taking the sqrt xfms it from noncentral chi^2 to rician
    %square to convert back
    sigMagNlm = NLmeansfilter((sigMag),t,f,hnl);
    sigMagNlmRician = NLmeansfilterRician(sqrt(sigMag),t,f,hnl,noiseVar).^2;
    %normalize by the noise variance so it becomes a noncentral chi^2 distributed r.v.
    %the magnitude^2 of a complex number with awgn is chi^2 with 2 degrees of freedom
    sigMagNlmNonCenChi2 = NLmeansfilterNonCenChi2(sigMag./noiseVar,t,f,hnl,2).*noiseVar;

    %switching for loops will probably make it more efficient, but less readable
    [pfa,pd] = prtScoreRoc(sigMag(:), sigTxLogical(:));
    [pfaNlm,pdNlm] = prtScoreRoc(sigMagNlm(:), sigTxLogical(:));
    [pfaNlmRician,pdNlmRician] = prtScoreRoc(sigMagNlmRician(:), sigTxLogical(:));
    [pfaNlmNonCenChi2,pdNlmNonCenChi2] = prtScoreRoc(sigMagNlmNonCenChi2(:), sigTxLogical(:));
    [pfaDwt,pdDwt] = prtScoreRoc(sigMagDwt(:), sigTxLogical(:));
    [pfaDwtPaper,pdDwtPaper] = prtScoreRoc(sigMagDwtPaper(:), sigTxLogical(:));

    if iter==0
        gammapd = pd;
        gammapfa = pfa;
        gammapdnlm = pdNlm;
        gammapfanlm = pfaNlm;
        gammapdnlmRician = pdNlmRician;
        gammapfanlmRician = pfaNlmRician;
        gammapdnlmNonCenChi2 = pdNlmNonCenChi2;
        gammapfanlmNonCenChi2 = pfaNlmNonCenChi2;

        gammapdDwt = pdDwt;
        gammapfaDwt = pfaDwt;

        gammapdDwtPaper = pdDwtPaper;
        gammapfaDwtPaper = pfaDwtPaper;

    else
        gammapd = gammapd + pd;
        gammapdnlm = gammapdnlm + pdNlm;
        gammapdnlmRician = gammapdnlmRician + pdNlmRician;
        gammapdnlmNonCenChi2 = gammapdnlmNonCenChi2 + pdNlmNonCenChi2;
        gammapfa = gammapfa + pfa;
        gammapfanlm = gammapfanlm + pfaNlm;
        gammapfanlmRician = gammapfanlmRician + pfaNlmRician;
        gammapfanlmNonCenChi2 = gammapfanlmNonCenChi2 + pfaNlmNonCenChi2;

        gammapdDwt = gammapdDwt + pdDwt;
        gammapfaDwt = gammapfaDwt + pfaDwt;

        gammapdDwtPaper = gammapdDwtPaper + pdDwtPaper;
        gammapfaDwtPaper = gammapfaDwtPaper + pfaDwtPaper;

    end
end




gammapd = gammapd./numIter;
gammapdnlm = gammapdnlm./numIter;
gammapdnlmRician = gammapdnlmRician./numIter;
gammapdnlmNonCenChi2 = gammapdnlmNonCenChi2./numIter;
gammapdDwt = gammapdDwt./numIter;
gammapdDwtPaper = gammapdDwtPaper./numIter;

gammapfa = gammapfa./numIter;
gammapfanlm = gammapfanlm./numIter;
gammapfanlmRician = gammapfanlmRician./numIter;
gammapfanlmNonCenChi2 = gammapfanlmNonCenChi2./numIter;
gammapfaDwt = gammapfaDwt./numIter;
gammapfaDwtPaper = gammapfaDwtPaper./numIter;

cmap =lines;
figure
subplot(2,2,[1 2])
plot(gammapfa,gammapd,'Color',cmap(1,:))
hold on
line_fewer_markers(gammapfanlm,gammapdnlm,10,'-+r', 'MarkerSize', 6,'Color',cmap(2,:))
line_fewer_markers(gammapfanlmRician,gammapdnlmRician,10,'-xy', 'MarkerSize', 6,'Color',cmap(3,:))
line_fewer_markers(gammapfanlmNonCenChi2,gammapdnlmNonCenChi2,10,'-om', 'MarkerSize', 6,'Color',cmap(4,:))
line_fewer_markers(gammapfaDwt,gammapdDwt,10,'-.g', '
    MarkerSize', 6,'Color',cmap(5,:))
line_fewer_markers(gammapfaDwtPaper,gammapdDwtPaper,10,'-*c'
    , 'MarkerSize', 6,'Color',cmap(6,:))
legend('Plain','With NLM','With Rician NLM','With Non-
    Central \chi^2 NLM','Soft Thresholded Wavelet','Hard
    Thresholded Wavelet','Location','southeast')
%legend('Plain','With NLM','With Rician NLM','With Non-
    Central \chi^2 NLM','Wavelet Paper')

xlabel('P_f_a')
ylabel('P_d')
title(sprintf('%s - ROC of Energy Detection With %.1f dB SNR
    ',figTitle,sinr));


subplot(2,2,3)
imagesc(sigMag);
sf = surf(T,F,10*log10(sigMag));
sf.EdgeColor = 'none';
axis tight
view(0,90)
title('Received Signal')
xlabel('Sample')
ylabel('Frequency (Hz)')

subplot(2,2,4)
sf = surf(T,F,10*log10(P));
sf.EdgeColor = 'none';
axis tight
view(0,90)
title('Transmitted Signal')
xlabel('Time (s)')
ylabel('Frequency (Hz)')

savefig(sprintf('%s/%s_roc_%ddB.fig',saveDir,savePrefix,sinr
    ))
saveas(gcf, sprintf('%s_eps/%s_roc_%ddB.eps',saveDir,
    savePrefix,sinr), 'eps2c')

result=1;
%{
figure
sf = surf(T,F,10*log10(sigMagDwt));
sf.EdgeColor = 'none';
```

```
axis tight
view(0,90)
title('Soft-Threshold Wavelet Denoising')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
savefig(sprintf('%s/%s_softThreshold_%ddB.fig',saveDir,
    savePrefix,sinr))

figure
sf = surf(T,F,10*log10(sigMagDwtPaper));
sf.EdgeColor = 'none';
axis tight
view(0,90)
title('Hard-Threshold Wavelet Denoising')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
savefig(sprintf('%s/%s_hardThreshold_%ddB.fig',saveDir,
    savePrefix,sinr))

figure
sf = surf(T,F,10*log10(sigMagNlm));
sf.EdgeColor = 'none';
axis tight
view(0,90)
title('Nonlocal Means')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
savefig(sprintf('%s/%s_nlm_%ddB.fig',saveDir,savePrefix,sinr
    ))

figure
sf = surf(T,F,10*log10(sigMagNlmRician));
sf.EdgeColor = 'none';
axis tight
view(0,90)
title('Rician Nonlocal Means')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
savefig(sprintf('%s/%s_ricianNlm_%ddB.fig',saveDir,
    savePrefix,sinr))

figure
sf = surf(T,F,10*log10(sigMagNlmNonCenChi2));
sf.EdgeColor = 'none';
axis tight
```

```
view(0,90)
title('Chi-Squared Nonlocal Means')
xlabel('Time (s)')
ylabel('Frequency (Hz)')
savefig(sprintf('%s/%s_nonCenChi2Nlm_%ddB.fig',saveDir,
   savePrefix,sinr))
%}

end
```

C Wavelet Hard Threshold Denoiser

```
function [denoisedSpec] = wavDenoisePaper(noisySignal,wname,
   dim)

if isempty(dim)
    dim = 1;
end

   if dim == 1
        [ca,cd] = dwt(noisySignal,wname);
        cd(:) = 0;
        denoisedSpec = idwt(ca,cd,wname);
   else
        %{
         [ca1,ch1,cv1,cd1] = dwt2(noisySignal,wname);
         ch1(:,:) = 0; cv1(:,:) = 0; cd1(:,:) = 0;
         [ca2,ch2,cv2,cd2] = dwt2(ca1,wname);
         ch2(:,:) = 0; cv2(:,:) = 0; cd2(:,:) = 0;
         [ca3,ch3,cv3,cd3] = dwt2(ca2,wname);
         ch3(:,:) = 0; cv3(:,:) = 0; cd3(:,:) = 0;
         %denoisedSpec = idwt2(ca3,ch3,cv3,cd3,wname);
         %denoisedSpec = idwt2(denoisedSpec,ch2,cv2,cd2,wname
            );
         %denoisedSpec = idwt2(denoisedSpec,ch1,cv1,cd1,wname
            );
         denoisedSpec = idwt2(ca1,ch1,cv1,cd1,wname);
         %}
         [c,s] = wavedec2(noisySignal,3,wname);
         %now set all vertical and horizontal subbands to 0
         vd3Start = s(1,1)*s(1,2)+s(2,1)*s(2,2)+1;
         vd3Len = 2*s(2,1)*s(2,2);
         c(vd3Start:vd3Start+vd3Len-1) = 0;

         vd2Start = vd3Start+vd3Len+s(3,1)*s(3,2);
         vd2Len = 2*s(3,1)*s(3,2);
         c(vd2Start:vd2Start+vd2Len-1) = 0;

         vd1Start = vd2Start+vd2Len+s(4,1)*s(4,2);
         vd1Len = 2*s(4,1)*s(4,2);
         c(vd1Start:vd1Start+vd1Len-1) = 0;


         denoisedSpec = waverec2(c,s,wname);

   end
end
```

D Wavelet Soft Threshold Denoiser

```
function [XC] = wavelet_denoise_builtin(X,wname,N)
% Returns a denoised image XC using matlab's builtin wavelet
    denoising
% u - image
% wname - wavelet name

[THR,SORH,KEEPAPP] = ddencmp('den','wv',X);
[XC,CXC,LXC,PERF0,PERFL2] = wdencmp('gbl',X,wname,N,THR,SORH
   ,KEEPAPP);
```

]

E Non-local Means Denoiser

```
function [output]=NLmeansfilter(input,t,f,h)

 %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

 %
 %  input: image to be filtered
 %  t: radio of search window
 %  f: radio of similarity window
 %  h: degree of filtering
 %
```

```
%  Author: Jose Vicente Manjon Herrera & Antoni Buades
%  Date: 09-03-2006
%
%  Implementation of the Non local filter proposed for A.
   Buades, B. Coll and J.M. Morel in
%  "A non-local algorithm for image denoising"
%
%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Size of the image
[m,n]=size(input);


% Memory for the output
output=zeros(m,n);

% Replicate the boundaries of the input image
input2 = padarray(input,[f f],'symmetric');

% Used kernel
kernel = make_kernel(f);
kernel = kernel / sum(sum(kernel));

h=h*h;

for i=1:m
for j=1:n

        i1 = i+ f;
        j1 = j+ f;

        W1= input2(i1-f:i1+f , j1-f:j1+f);

        wmax=0;
        average=0;
        sweight=0;

        rmin = max(i1-t,f+1);
        rmax = min(i1+t,m+f);
        smin = max(j1-t,f+1);
        smax = min(j1+t,n+f);

        for r=rmin:1:rmax
```

```
         for s=smin:1:smax

                if(r==i1 && s==j1) continue; end;

                W2= input2(r-f:r+f , s-f:s+f);

                d = sum(sum(kernel.*(W1-W2).*(W1-W2)));

                w=exp(-d/h);

                if w>wmax
                    wmax=w;
                end

                sweight = sweight + w;
                average = average + w*input2(r,s);
         end
         end

        average = average + wmax*input2(i1,j1);
        sweight = sweight + wmax;

        if sweight > 0
            output(i,j) = average / sweight;
        else
            output(i,j) = input(i,j);
        end
 end
 end

function [kernel] = make_kernel(f)

kernel=zeros(2*f+1,2*f+1);
for d=1:f
  value= 1 / (2*d+1)^2 ;
  for i=-d:d
  for j=-d:d
    kernel(f+1-i,f+1-j)= kernel(f+1-i,f+1-j) + value ;
  end
  end
end
kernel = kernel ./ f;
```

F Rician Non-local Means Denoiser

```
function [output]=NLmeansfilterRician(input,t,f,h,nVar)

%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%  input: image to be filtered
%  t: radio of search window
%  f: radio of similarity window
%  h: degree of filtering
%
%  Author: Jose Vicente Manjon Herrera & Antoni Buades
%  Date: 09-03-2006
%
%  Implementation of the Non local filter proposed for A.
   Buades, B. Coll and J.M. Morel in
%  "A non-local algorithm for image denoising"
%
%
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Size of the image
[m,n]=size(input);


% Memory for the output
output=zeros(m,n);

% Replicate the boundaries of the input image
input2 = padarray(input,[f f],'symmetric');

% Used kernel -- gaussian kernel
kernel = make_kernel(f);
%for now make the kernel ones(f) normalized
%kernel=ones(2*f+1,2*f+1);

kernel = kernel / sum(sum(kernel));

%h=h*h;


```

```
%i,j are the pixels we are creating blocks for
for i=1:m
    for j=1:n

        %i1,j1 are the center of the block
        i1 = i+ f;
        j1 = j+ f;

        %the base block has similarity window of radius f and centered at (i1,j1)
        W1= input2(i1-f:i1+f , j1-f:j1+f);

        wmax=0;
        average=0;
        sweight=0;

        %setting up the search window
        %search window will be for blocks centered at (rmin,smin)->(rmax,smax)
        rmin = max(i1-t,f+1);
        rmax = min(i1+t,m+f);
        smin = max(j1-t,f+1);
        smax = min(j1+t,n+f);

        for r=rmin:1:rmax
            for s=smin:1:smax

                if(r==i1 && s==j1) continue; end;

                %window of comparison
                W2= input2(r-f:r+f , s-f:s+f);

                %distance metric with a drop off based on the kernel
                %d = sum(sum(kernel.*(W1-W2).*(W1-W2)));
                d = sqrt(sum(sum(kernel.*(W1-W2).*(W1-W2))));

                %tap weights for the filter with drop off h
                w=exp(-d/h);

                %if the weight for the block is greater than the greatest weight so far, update
                if w>wmax
                    wmax=w;
                end

                %sum weights for normalization later
                sweight = sweight + w;
                %average block seen so far
                %average = average + w*input2(r,s);
                %equation 3
                average = average + w*input2(r,s).^2;
            end
        end

        %update average and sweight with weight for the current block being searched
        average = average + wmax*input2(i1,j1).^2;
        sweight = sweight + wmax;

        %normalize the output block
        if sweight > 0
            %output(i,j) = sqrt(max(average ./ sweight - 2.*nVar);
            output(i,j) = sqrt(average ./ sweight - 2.*nVar);

            %abs()^2 makes it noncentral chi^2
        else
            output(i,j) = input(i,j);
        end

    end
end

function [kernel] = make_kernel(f)

kernel=zeros(2*f+1,2*f+1);
for d=1:f
    value= 1 / (2*d+1)^2 ;
    for i=-d:d
        for j=-d:d
            kernel(f+1-i,f+1-j)= kernel(f+1-i,f+1-j) + value;
        end
    end
end
kernel = kernel ./ f;
```

G Noncentral χ^2 Non-local Means Denoiser

```
function [output]=NLmeansfilterNonCenChi2(input,t,f,h,k)

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%
%  input: image to be filtered
%  t: radio of search window
%  f: radio of similarity window
%  h: degree of filtering
%  k: chi^2 degree parameter
%
% BASED ON:
%  Author: Jose Vicente Manjon Herrera & Antoni Buades
%  Date: 09-03-2006
%
%  Implementation of the Non local filter proposed for A. Buades, B. Coll and J.M. Morel in
%  "A non-local algorithm for image denoising"
%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Size of the image
[m,n]=size(input);


% Memory for the output
output=zeros(m,n);

% Replicate the boundaries of the input image
input2 = padarray(input,[f f],'symmetric');

% Used kernel -- gaussian kernel
kernel = make_kernel(f);
%for now make the kernel ones(f) normalized
%kernel=ones(2*f+1,2*f+1);

kernel = kernel / sum(sum(kernel));

%h=h*h;
```

```


%i,j are the pixels we are creating blocks for
for i=1:m
    for j=1:n

        %i1,j1 are the center of the block
        i1 = i+ f;
        j1 = j+ f;

        %the base block has similarity window of radius f
           and centered at (i1,j1)
        W1= input2(i1-f:i1+f , j1-f:j1+f);

        wmax=0;
        average=0;
        sweight=0;

        %setting up the search window
        %search window will be for blocks centered at (rmin,
           smin)->(rmax,smax)
        rmin = max(i1-t,f+1);
        rmax = min(i1+t,m+f);
        smin = max(j1-t,f+1);
        smax = min(j1+t,n+f);

        for r=rmin:1:rmax
            for s=smin:1:smax

                if(r==i1 && s==j1) continue; end;

                %window of comparison
                W2= input2(r-f:r+f , s-f:s+f);

                %distance metric with a drop off based on
                   the kernel
                %d = sum(sum(kernel.*(W1-W2).*(W1-W2)));
                d = sum(sum(kernel.*(W1-W2).*(W1-W2)));

                %tap weights for the filter with drop off h
                w=exp(-d/h);

                %if the weight for the block is greater than
                    the greatest weight so far, update
                if w>wmax
```

```matlab
                    wmax=w;
                end

                %sum weights for normalization later
                sweight = sweight + w;
                %average block seen so far
                %average = average + w*input2(r,s);
                %equation 3
                average = average + w*input2(r,s);
            end
        end

        %update average and sweight with weight for the current block being searched
        average = average + wmax*input2(i1,j1);
        sweight = sweight + wmax;

        %normalize the output block
        if sweight > 0
            output(i,j) = (average ./ sweight - k);
            %abs()^2 makes it noncentral chi^2
        else
            output(i,j) = input(i,j);
        end
    end
end

function [kernel] = make_kernel(f)

kernel=zeros(2*f+1,2*f+1);
for d=1:f
    value= 1 / (2*d+1)^2 ;
    for i=-d:d
        for j=-d:d
            kernel(f+1-i,f+1-j)= kernel(f+1-i,f+1-j) + value ;
        end
    end
end
kernel = kernel ./ f;
```