Department of Electronic Engineering

And Computer Engineering

Technical University of Crete

DIPLOMA THESIS

by

Oikonomakou Valentina

Subject:

Spectrogram segmentation-based tracking of a frequency-hopped signal

Abstract

Frequency hopped complex sinusoids are most appealing due to their resistance to interference and the difficulty of being intercepted. They are extensively used in wireless and in military communications. Therefore, the problem of tracking the frequency of a frequency hopped signal is widely spread and quite interesting. In this thesis we propose a spectrogram segmentation-based approach using edge detection. In the end of the thesis, we present simulation runs of the algorithm, for which we used a stochastic state-space model, and discuss its performance compared to two alternative approaches: one using particle filtering tools and one using the spectrogram.

Keywords: Frequency hopping, spectral analysis, time-frequency distribution, edge detection, particle filtering

Advisor: Nikos D. Sidiropoulos

Chapter 1

The first chapter functions as an introduction to the entire thesis. It provides some parts of the theory background needed to better comprehend the problem that is stated in the thesis. Furthermore, it contains information about the stochastic state-space model that was used and about the first steps that led to the development of the spectrogram segmentation-based approach.

1.1 Introduction

Frequency–hopping spread spectrum (FHSS) is a modulation technique by which SS signals rapidly switch frequencies using a random sequence known to both receiver and transmitter. Spread spectrum transmission is widely used due to three main advantages over fixed–frequency transmission: SS signals are highly resistant to interference, difficult to intercept and can share bandwidth with many types of conventional transmissions with minimal interference. FHSS transmission is often used in wireless and military communications for the purpose of mitigating interference and for its resistance to jamming.

Tracking the frequency of a FHSS signal can be rather interesting considering it is a problem that arises in various applications. For example, in speech processing, tracking formant frequencies is quite common. In wireless communications, the receiver could have no prior knowledge of the hopping pattern, or could just be out of sync with the transmitter's generator.

In this thesis we present a spectrogram segmentation–based approach for tracking a frequency-hopped (FH) signal**.** Later on, it is compared to two alternative algorithms: one using spectrogram and one using particle filtering (PF). The data model all the above algorithms have worked on is actually a simple and stochastic state-space model.

1.2 Data Model

The data model that was used by the algorithms, as they are presented in the thesis, was first proposed in [6] (also see chapter 3, section 3.2.1) and it is in fact a non-linear non-Gaussian stochastic state-space model of a frequency-hopped complex sinusoid. It is also presented here for simplicity reasons.

Let $x_k := [\omega_k, A_k]^T$, where $\omega_k \in [-\pi, \pi)$ and $A_k \in C$ denote the frequency and amplitude at time k. Let $\mathbf{u}_k := [b_k, \tilde{\omega}_k, \tilde{A}_k]^T$ denote an auxiliary sequence of independent and identically distributed (i.i.d) vectors with independent components and the following marginal statistics: b_k is a binary random variable with probability $P(b_k = 1) = h$; $\tilde{\omega}_k$ is uniformly distributed over $[-\pi, \pi)$, denoted $U([-\pi, \pi))$; and $\tilde{A}_k$ is $CN(0, \sigma^2_A)$, i.e., complex circular Gaussian of variance σ^2_A. Then

$$x_k = f(x_{k-1}, u_k) = \begin{cases} x_{k-1} & ,u_k(1)=0 \\ [u_k(2), u_k(3)]^T & ,u_k(1)=1 \end{cases}$$

$$= \begin{cases} x_{k-1} & \text{, w.p. } 1-h \\ [U([-\pi, \pi)), \mathrm{CN}(0, \sigma^2_A)]^T & \text{, w.p. } h \end{cases}$$

$$y_k = x_k(2) e^{j x_k(1) k} + v_k,$$

where v_k denotes i.i.d $CN(0, \sigma^2_n)$ measurement noise, and $u_k(1)$ the hop variable.

It is important to point out the fact that the above state space formulation *does not* assume that frequency hops periodically, in contrast to traditional models of frequency hopping. It is_motivated by one main consideration: it is often required, especially in military communications, to intentionally jitter the hopping time in order to reduce the risk of data interception (anti-jamming). The expected number of hops, according to the presented *probabilistic* model, over a long observation interval T is hT, where h denotes the hopping probability. Probabilistic modeling is more accurate if the hop period is inaccurate.

1.3 The Idea

This work is based on the distribution approach of S. Barbarossa and A. Scaglione [1] for tracking a FH signal. They proposed a non - parametric algorithm based on the time-frequency representation of the observed signal. In particular, they suggested the Wigner-Ville distribution (WVD) as the time frequency distribution for the algorithm.

The WVD can be interpreted as the time-frequency energy distribution of a signal. On the whole it gives better temporal and frequency resolution, relative to other time-frequency distributions, such as the spectrogram, at the expense, though, of computation

complexity (the presence of cross terms also adds up to it) and the introduction of negative values, which would correspond in negative energy (this is not physically possible and represents a significant defect of this method). However, these problems are well known and there are ways to compensate them. WVD is also a member of the large family of time-frequency distributions known as Cohen's class and presents a number of desirable mathematical properties. For example, it is always real-valued.

Let x(n) denote the sequence of samples of the observed signal, where n = 0, …, N-1 (N samples). The estimation algorithm they proposed consists of the following steps:

- Computation of the discrete WVD (Wx) of the signal x

$$W_x(x,k) = 2\sum_i x^*(n-i)x(n+i)e^{-j4\pi\frac{ik}{N}}$$

- Computation of the smoothed WVD (Ws(x,k)) using a lowpass filter
- Computation of the maxima of Ws(x,k) for each n (y(n))
- *Estimation of T (= number of samples per hop) as the period of y(n)*
- Estimation of the number of segments (N_S) falling within the observation interval ($Ns=\lceil N/T \rceil$)
- Computation of the function

$$P_l(n_0,k) = \sum_{n=n_0+lN_h}^{n_0+(l+1)N_h} W_x(n,k),$$

for $l = 1,...,N_s - 1$ and $n_0 = -T/2,...,T/2$

- Estimation of the time offset n_0 as

$$\hat{n_0} = \arg\max_{n_0}\{\sum_{l=1}^{N_s-1}\sum_{k=0}^{N-1} P_l(n_0,k)\}$$

- Estimation of the frequency f_l as:

$$f_l = \arg\max_{x_k} \left\{ \sum_{\hat{n}_o + lN_h}^{\hat{n}_o + (l+1)N_h} W_x(n,k) \right\} / 2N$$

for l=0, 1, ..., (T-1)

It is important to note that Barbarossa and Scaglione's estimation algorithm applies for models where the frequency hops periodically (see fourth step above). The probabilistic model we used, though, does not assume that. So, the algorithm we present in this thesis, in order to estimate the hop timing, uses the technique of edge detection.

1.4 The algorithm

The proposed algorithm consists of fours basic steps: 1) Calculation of the spectrogram as the time-frequency distribution of the observed signal, 2) Calculation of a vector containing the frequency mode (location of peak of the spectrogram as a function of time), 3) Application of the edge detection technique in order to acquire the times the frequency hops, 4) Use of the periodogram method in order to estimate the signal's frequency for each of the time intervals, resulting from using edge detection. The algorithm is thoroughly explained step by step in the following chapter.

Notice that we could estimate the frequency of the signal just by using the data obtained from the spectrogram of the signal. The purpose of the edge detection is to spot the exact time the frequency hops. At the spectrogram's estimation the hopping occurs gradually (ramp). For example, if the true hop occurs at time t = 50, the spectrogram's frequency hop estimation could arise within a time interval: 48-53. Now, the edge detector, theoretically at least, locates the true time the frequency hops. This, along with the fact that by splitting the signal into segments, within which the signal's frequency is fixed, and *then* estimating the frequency at each dwell, the estimation is much more accurate.

Chapter 2

This chapter is actually the main one of the thesis. It describes the proposed algorithm step by step including the theory for each and every one of them. In particular, every section describes a step of the algorithm: 4 steps, 4 sections, and each one cites information concerning the corresponding theory.

2.1 Time - Frequency Distribution

In order to choose the best time-frequency distribution for the given data model, both WVD and spectrogram were tested for many variations of white Gaussian noise. The spectrogram offered much better results. The spectrogram is the result of calculating the power spectrum of window frames of a signal as it changes over time. In order to calculate the power spectrum, it uses the periodogram power spectrum estimation via Fast Fourier Transform (FFT, section 2.1.1).

More specifically, let T denote the length of the window frames and *nfft* (nfft >= T) denote the number of frequencies at which the Fourier Transform (FT) is computed. Usually, the frames overlap by a number of *n* samples and the process begins after acquiring at least T samples. Therefore, it computes *one* value for every (T – n) samples and it uses future samples to estimate the frequency at a present time (non-causal windowing). Though that is the most common format of the spectrogram, we used a running - window spectrogram (RWS). The choice was made based on the results that were obtained after performing a number of simulations on both of them.

The difference between the two lies in the samples processed at each frame. In the running – window spectrogram that was used, the process begins as soon as the first sample is acquired and the empty (T – 1) slots are filled with zeroes. The process continues respectively until (T + 1) samples are obtained. After that, the frames contain T samples and the overlapping number is (T – 1), similarly to the usual format (for n = T-1). Hence, it computes one value for each sample and without using any future samples (causal windowing).

2.1.1 The Periodogram

As it has already been mentioned, our data model is random, meaning that its variation in the future can not be known exactly. A random signal can be described as a random sequence which consists of a set of possible realizations, each of which has some associated probability to occur. The realizations, viewed as discrete-time sequences (samples), *do not* have finite energy *but* they usually have finite average power. Therefore, they can be described by an average *power spectral density* (PSD) [3].

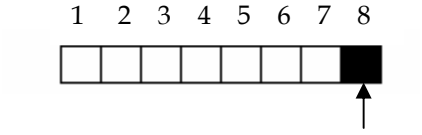
The PSD estimator we used is called *periodogram spectral estimator*: if we have a signal that is exponential of amplitude A embedded in AWGN: $y_k = Ae^{i\omega k} + w_k$ then, the frequency at which the periodogram, which is defined as the squared-magnitude of the DTFT of y_k:

$$\hat{\phi}_p = \frac{1}{N}\left|\sum_{k=1}^{N} y_k(t)e^{-i\omega k}\right|^2,$$

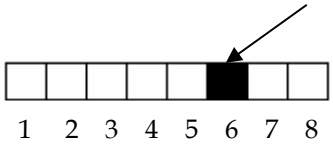
peaks, is the estimated frequency **ω** of the signal.

2.2 Maximum Power Spectral Density

After the computation of the time frequency distribution, we obtain a vector containing for each sample the frequency at which the PSD is maximum. As it has already been mentioned above, *nfft* is the number of frequencies at which the FFT is calculated. Additionally, due to the fact that in the beginning of the spectrogram's frequency estimation there are not enough temporal samples, the estimation seems to delay for a couple of time samples. So, we discard the estimated frequencies of the first two samples and extend the last one. In other words, we shift the estimation sequence to the left and fill in the last two empty spots with the value of the last estimated frequency. That, though, is as if we used non-causal windowing for the spectrogram estimation because in fact, we use future samples to accurately detect an edge in the "present": imagine, for example, the causal–window consisting of 8 samples that are used for the frequency estimation of the 8th sample:



Next, we shift the estimation two slots to the left as if we had a non-causal window of length 8 that we used for the frequency estimation of the 6th sample:



2.3 Edge Detection

2.3.1 Introduction

Edge detection is one of the most commonly used operations in image processing since edges form the outline of an object. This can become clearer if we think an edge as the boundary between an object and the background and among overlapping objects. This means that if the edges can be accurately identified, all the objects can be traced and basic properties such as area, perimeter and shape can be measured. Considering, then, that image analysis involves the identification and classification of objects in an image, edge detection is an essential tool.

2.3.2 Canny Edge Detector

2.3.2.a Introduction

One of the best advanced edge detection techniques is the one introduced by John Canny in 1986 [5]. Canny defined a set of goals for an edge detector and described an optimal method for achieving them. These are:

1. *Good Detection* - The edge detector should, ideally, respond only to real edges and should find all of them.

2. *Good Localization* - The distance between the edge pixels as found by the edge detector and the actual edge should be as small as possible.

3. *Response* - The edge detector should not identify multiple edge pixels where only a single one exists.

Canny assumed a step edge subject to white Gaussian noise. The edge detector was assumed to be a convolution filter *f* which would smooth the noise and locate the edges. The problem was to identify the one filter (f(x)) that optimizes the three edge detection criteria.

As the input signal (G(x)) to the edge detector, Canny used a step edge of amplitude A bathed in white Gaussian noise:

$$G(x) = Au(x) + n(x)$$

where u(x) is the unit step function which is defined as follows,

$$u(x) = \begin{cases} 0, & \text{for } x<0 \\ 1, & \text{for } x\geq 0 \end{cases}$$

Canny expressed the first criterion as the maximization of the SNR which was defined as the ratio of the output response to the step only to the square root of the mean squared noise response.

Let x_0 be the centre of the step edge, then the output $O(x_0)$ of the convolution with the input signal G(x) is given by

$$O(x_0) = \int_{-\infty}^{+\infty} G(x) f(x_0 - x)dx$$

The output response due to the step only for x_0=0 is

$$O_s(x_0) = \int_{-\infty}^{+\infty} f(x)Au(-x)dx = A\int_{-\infty}^{0} f(x)dx$$

The mean squared response to noise is

$$E\left[\int_{-\infty}^{+\infty} f(x)n(-x)dx\right]^2 = E\left[\int_{-\infty}^{+\infty} f^2(x)n^2(-x)dx\right] = n_0^2\int_{-\infty}^{+\infty} f^2(x)dx$$

where $n_0^2 = \mathrm{E}[n^2(x)]$ (noise input variance).

Therefore the SNR can be expressed as

$$SNR = \frac{A\int_{-\infty}^{0} f(x)dx}{n_0\sqrt{\int_{-\infty}^{+\infty} f^2(x)dx}}$$

The *Localization* value was defined as the reciprocal of the distance of the located edge from the true edge and should be as large as possible, too. Canny chose to mark edges at the maximum of the output $O(x_0)$ and in order to find x_0, we set

$$O'(x_0) = \frac{d}{dx}\int_{-\infty}^{+\infty} f(x)G(x_0 - x)dx \overset{\text{differantiation theorem for convolution}}{=} \int_{-\infty}^{+\infty} f'(x)G(x_0 - x)dx = 0 \qquad (1)$$

Once again we use the linearity for convolution in order to get the derivative of the output to the step only ($O'_s(x_0)$)

$$O'_s(x_0) = \int_{-\infty}^{+\infty} f'(x)Au(x_0 - x)dx = \int_{-\infty}^{x_0} Af'(x)dx = Af(x_0)$$

and the derivative of the output due to noise ($O'_n(x_0)$) which will be a Gaussian random variable with mean zero and variance

$$E\left[O'^2_n(x)\right] = n_0^2\int_{-\infty}^{+\infty} f'^2(x)dx \qquad (2)$$

By adding the constraint for the f(x) to be antisymmetric and using the Taylor expansion we get:

$$O'_s(x_0) = Af(x_0) \approx x_0 Af'(0) \qquad (3)$$

Finally, we have

$$(1) \Rightarrow O'(x_0) = O'_s(x_0) + O'_n(x_0) = 0$$
$$\Rightarrow O'_s(x_0) = -O'_n(x_0)$$
$$\Rightarrow E\left[O'^2_s(x_0)\right] = E\left[O'^2_n(x_0)\right]$$
$$\overset{(2),(3)}{\Rightarrow} E\left[x_0^2\right] \approx \frac{\int_{-\infty}^{+\infty} n_0^2 f'^2(x)dx}{A^2 f'^2(0)}$$

where $\sqrt{E\left[x_0^2\right]}$ is the standard deviation of the distance of the located edge from the true edge.

So, the *Good Localization* criterion is expressed as the maximization of:

$$Localization = \frac{A}{n_0} \frac{\left|f'(0)\right|}{\sqrt{\int_{-\infty}^{+\infty} f'^2(x)dx}}$$

Thus, Canny tried to find the filter that maximizes the product *SNR*Localization* which is both amplitude and scale independent [7]. However, the edge detector that is derived from this combination of the first two criteria provides multiple responses to the true edge (multiple maxima which are close to each other) which was defined as the centre of the step edge. That is due to interaction of the responses (to a noisy step edge) at numerous points near the edge.

Hence, Canny added an extra constraint: the value of the distance between adjoining maxima in the filtered response or zero-crossings of their derivatives will be twice the value of the mean distance x_{zc}: A result due to Rice (1944, 1945) states that the average distance between zero-crossings of the response of a function g to Gaussian noise is

$$x_{ave} = \pi\left(\frac{-R(0)}{R''(0)}\right)^{\frac{1}{2}}$$

where R(τ) denotes the autocorrelation of the function g.

So, since $R(0) = \int_{-\infty}^{+\infty} g^2(x)dx$ and $R''(0) = -\int_{-\infty}^{+\infty} g'^2(x)dx$ we get that

$$x_{zc} = \pi \left(\frac{\int_{-\infty}^{\infty} f'^2(x)dx}{\int_{-\infty}^{\infty} f''^2(x)dx} \right)^{\frac{1}{2}}$$

The result of the maximization of the product *SNR*Localization* subject to the multiple response constraint, though, was too complex to be solved analytically. But the *first derivative of a Gaussian* function turned out to be a much satisfying approximation.

2.3.2.6 The Canny edge detection algorithm

Recall, that in the presented algorithm we need the Canny edge detector to find when the frequency hops and that as an input to the algorithm we use the vector containing the frequencies obtained from the Spectrogram frequency distribution. Before proceeding, though, to the edge detection, we need to smooth the input by performing a multidimensional filtering using convolution with a one-dimensional Gaussian mask. In particular, we used the following Gaussian:

$$G_w[k+1] = e^{-\frac{1}{2}\left(a\frac{k}{N/2}\right)^2}$$

where N is the length of the window, k = [-N/2:N/2] and α is the reciprocal of the standard deviation. They both are parameters to the edge detector. The above Gaussian was chosen through experimental simulations and was found in the signal processing Matlab toolbox.

Now that we have smoothed the input (I_s), using the above Gaussian mask, we may continue to the edge detection.

The **first step** would be to perform a multidimensional convolution of I_s with the derivative of a Gaussian Mask. Let σ be the standard deviation of this Gaussian: this is also a parameter to the edge detector and it determines the length of the Gaussian mask.

Specifically, we arbitrarily set a number of possible widths **w** (i.e. 1:30) and calculate the corresponding values of the following Gaussian function [4]:

$$\text{Gaussian}(w) = e^{-\left(\frac{\mathbf{w}^2}{2\sigma^2}\right)}$$

The greatest value of width w**,** that gives a value larger than a certain value (i.e. 0.001), is the number used to create a vector **y**: **y** = [-w : w]. So, we obtain the following one-dimensional Gaussian mask:

$$f(\mathbf{y}) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{\mathbf{y}^2}{2\sigma^2}\right)}$$

The derivative of the above Gaussian $f'(\mathbf{y}) = \left(-\frac{\mathbf{y}}{\sigma^2}\right) f(\mathbf{y})$ is the one that is used for the convolution (first step of the edge detector). Let I_s^c denote the result of that convolution.

Next, we must simply estimate the magnitude M of I_s^c at each point. Now, assuming that in fact the value of the magnitude at a certain point is large, if it is an edge, and smaller, if it is not, the **next step** is a *non-maximum suppression* step, where points that are not local maxima are removed: the points presumed to be edges are the ones with a value greater than the value of their adjacent pixels (above and below). So, the non-maximum suppression step provides a set of *possible* edges.

As a **last step**, Canny (in image processing (2D)) suggested thresholding using *hysteresis* rather than simply selecting a threshold value to apply everywhere. Hysteresis thresholding uses a high threshold T_{high} and a low threshold T_{low}. Both thresholds are parameters to the algorithm. All pixels that have a value* greater than T_{high} are presumed to be edge pixels (strong edges) and are marked as such immediately. The pixels with a value larger than T_{low} are possible edges (weak edges) and are marked as real edges only if they are connected to a strong edge, directly or through a chain of weak edges. For example, imagine 3 pixels in a row. Let the first two pixels be weak edges and the third one to be a strong edge. The second one is, in the end, marked as an edge because it is directly connected to a strong edge (third pixel). The first one is also

marked as an edge because it is connected to a weak edge (second pixel) which is next to a strong one (third pixel, connecting through a chain). The result the Canny edge detector finally presents is an array of aces and zeroes (black and white image), where the ones indicate the edges (white) and the zeroes (black) the background.

In the presented algorithm we work in one dimension. Plus, in the presented application of the Canny edge detector, there is no way there will be adjoining edges. Thus, we do not need hysteresis thresholding and we should just apply one threshold value everywhere, instead. That though did not give good results (see section 3.1, chapter 3). That is why we chose to use a threshold, to eliminate most of the non-edges and the weak edges, *plus* a rejection of the smaller dwells. This rejection was decided because an estimated small dwell is not true, most of the time, and when there actually is one it is really hard for the algorithm to accurately detect it. So, we group the shorter ones in longer ones. This process consists of two parts: Let **E** denote the vector with the maximum magnitudes obtained after thresholding. First, we nullify the first f and the last l values of vector E. It is a common phenomenon for the edge detection algorithm to detect false edges in the beginning and in the end of the observed sequence, especially as the noise variance grows. Second, when an edge is detected, if another one is detected in less then t samples, we only keep the first one. All variables f, l and t are parameters to the algorithm.

Before we proceed, allow us to point out that, for simplicity reasons, the parameters N, α and σ could be set to the following values respectively: 8, 6 and 0.7 (these values were used for the simulations that are presented later on). They offer satisfying results for a great set of different cases.

2.4 Final Frequency Estimation

In the final step, we divide the signal into segments, correspondingly to the edges detected, and process them separately. In order to estimate the frequency of the signal within each dwell, we use the periodogram method. If the dwells are large enough and, therefore, there are enough samples, the estimation is very accurate.

Chapter 3

In this chapter we present a variety of simulations for the proposed algorithm and for the algorithm that uses particle filtering (PF) tools. To start with, we comment on each algorithm's performance separately. For the particle filtering we provide, in addition, some general information on PF and on the corresponding PF algorithm [6].

3.1 Proposed Algorithm Simulations

We will now present the simulation results for the proposed algorithm. The following table contains the Root Mean Square Error (RMSE) of the frequency estimation of the algorithm for a variety of White Gaussian noise. The parameters used for the simulations are the following: h = 0.01, σ^2_A = 1, T = 100. The number of Monte-Carlo (MC) simulations is 300. In addition to the RMSE we present the mean number of estimated edges and the hop-range compared to the original ones in order to estimate the edge detector's performance. Furthermore, so as to support our choice of edge detecting (section 2.3.3, chapter 2), we present the values of the RMSE, the mean and the range of the number of estimated hops for the case where just one threshold value is used instead.

In the following table, the case where one threshold value is used is divided into two different sets of results. The first one corresponds to the results of the algorithm when trying to reach the minimum RMSE, and the second one corresponds to the results when the aim was to estimate, by average, the same number of hops as the one provided by the algorithm for the proposed edge detection.

σ_n^2=0.2, *True Mean #Hops = 0.9400, True Hop-Range = 0 - 4*	**RMSE**	**Estimated Mean #Hops**	**Estimated Hop-Range**	**Time for 1 simulation (sec)**
Proposed edge detector	0.1841	0.9533	0 - 5	0.0695
Single threshold value	0.3163	3.1100	0 - 24	0.0695
σ_n^2=0.3, *True Mean #Hops = 1.0200, True Hop-Range = 0 - 5*	**RMSE**	**Estimated Mean #Hops**	**Estimated Hop-Range**	**Time for 1 simulation (sec)**
Proposed edge detector	0.2543	1.0800	0 - 5	0.1654
Single threshold value	0.4157	3.4667	0 - 28	0.1648
σ_n^2=0.5, *True Mean #Hops = 0.8700, True Hop-Range = 0 – 4*	**RMSE**	**Estimated Mean #Hops**	**Estimated Hop-Range**	**Time for 1 simulation (sec)**
Proposed edge detector	0.2744	1.4600	0 - 5	0.1187
Single threshold value	0.4826	1.6800	0 - 14	0.1587
σ_n^2=0.8, *True Mean #Hops = 0.8933, True Hop-Range = 0 - 5*	**RMSE**	**Estimated Mean #Hops**	**Estimated Hop-Range**	**Time for 1 simulation (sec)**
Proposed edge detector	0.3668	2.3300	0 - 5	0.1464
Single threshold value	0.5819	2.3600	0 - 15	0.1320

Let us, first, comment on the performance of the proposed edge detector (section 2.3.3, chapter 2) compared to the one using a single threshold value. As the above simulations of the presented algorithm show, the RMSE we obtained when we used the proposed detection method, is *much* smaller, and rather good in general for σ_n^2=0.2, than the RMSE obtained from the algorithm when we used only one threshold value.

Another advantage of the detection method we propose, is the fact that it can provide, at least for white Gaussian noise with a standard deviation less than 0.5, a lower RMSE than the single threshold value, for approximately the same mean number of estimated hops as the original one. Certainly, as the noise power grows, the number of estimated "possible" edges (local maxima) increases, making it harder and harder for the detector to discern the true hops from the false ones. That is why for σ_n^2=0.8, for example, the mean of estimated edges, even when we used the proposed edge detection, is more than $2^{1/2}$ times the original one. In other words, we let many false edges pass for true ones, as that offered a more desirable RMSE. Additionally, the range of the number of estimated hops that is provided by the algorithm when the proposed edge detection is used, always remains close to the true hop-range: which is a good thing considering it hardly estimates more edges than there could actually be there. On the contrary, when a single threshold value is used, the corresponding hop-range is much greater than the true one. Even in case where the mean number of estimated hops is close to the original one, the hop-range is much wider: which practically means that the algorithm either estimates too few edges or too many!

Furthermore, the proposed edge detection does not add up to the complexity of the algorithm, as shown from the time needed for a single data sequence to be processed. Plus, it helps the algorithm to be a bit more resistant to noise. The difference between the RMSE of the simulations for σ_n^2=0.2 and σ_n^2=0.8 and for the proposed edge detection, is approximately 0.18 when for the single threshold value is around 0.27.

An original edge might be missed due to two main reasons: First, if the frequency hops to a value close to the current one or a hop occurs in a short period of time, the edge is not that strong and it is hard to even pass through the non-maximum suppression process (not a local maximum). This, though, does not, usually, add up significantly to the overall RMSE. Second, if within a data sequence there are, for example, 3 edges (four dwells), there is a high probability that one, at least, edge will have a magnitude of a much lower value than, not only the magnitude values of the remaining edges, but of false ones, too. If false edges have large magnitude values, it is usually due to noise. That is why in the proposed thresholding we mostly use a relatively low threshold value: to let more original edges pass through. And that is why we group the shorter dwells into longer ones: to eliminate more false edges. Small dwells are frequently the consequence of the high magnitude values of *false* edges. This, though, sometimes costs in RMSE (if a true small dwell is missed) *but* we had to find a suitable trade-off!

Considering that a RMSE up to 0.15 - 0.2 is satisfying, the proposed algorithm for white Gaussian noise with a standard deviation σ_n^2 = 0.2, noise that is not negligible at all,

works relatively good. For a standard deviation of $\sigma_n^2 = 0.3$ there is a noticeable increase in the RMSE that remains nearly the same for up to $\sigma_n^2 = 0.5$. For $\sigma_n^2 = 0.8$ the increase of the RMSE is significant. That happens probably because as the noise power grows: the spectrogram produces a less accurate estimation, more intense false hops are produced plus the periodogram in the final step does not always estimate the frequency accurately. Even so, the RMSE is less than one would expect it to be.

Additionally, the presented algorithm is simple and fast. The execution time for a single simulation of 100 temporal samples is less than 0.1 sec for $\sigma_n^2=0.2$ and less than 0.2 sec for all the presented cases.

The following figures show the performance of the algorithm in terms of the RMSE for $\sigma_n^2=0.2, 0.3, 0.5, 0.8$ and a few simulation runs of the algorithm.

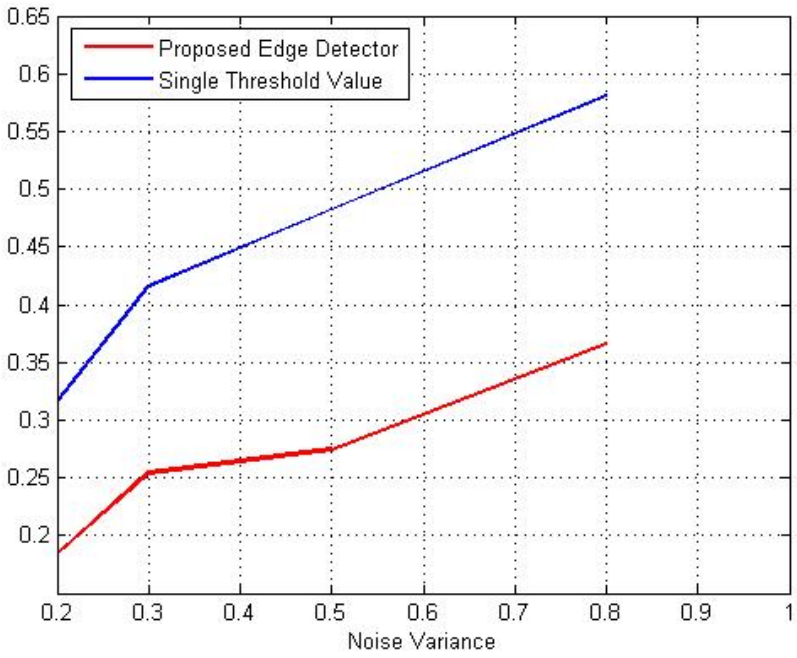


Figure 1: *Performance of the algorithm in terms of the RMSE using the proposed edge detector vs. using a single threshold value*

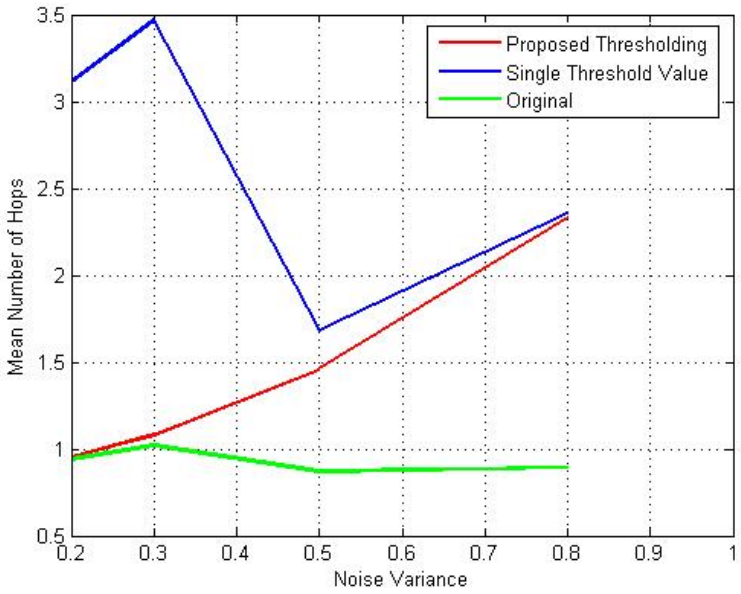


Figure 2: *Performance of the algorithm in terms of the estimated mean number of hops using the proposed thresholding vs. using a single threshold value*

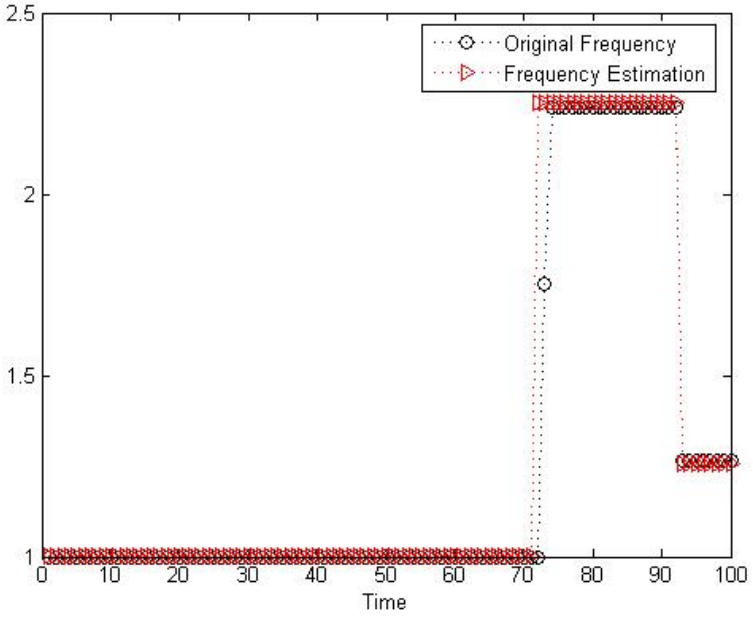


Figure 3: *Typical sample simulation run of the proposed algorithm*

Now, the following table contains the results of some more simulations of the algorithm for a higher hopping probability and for data sequences consisting of more than 100 temporal samples. The parameters of the simulations are: σ_n^2=0.2, σ_A^2=1, MC = 300.

	RMSE	**Estimated Mean #Hops**	**Estimated Hop-Range**	**Time for 1 simulation (sec)**
***h = 0.01*, T = 100** **True Mean #Hops = 0.9400** **True Hop-Range = 0 - 5**	0.1841	0.9533	0 - 5	0.0695
***h = 0.02*, T = 100** **True Mean #Hops = 2.0133** **True Hop-Range = 0 - 7**	0.4041	2.0633	0 - 8	0.0702
***h = 0.03*, T = 100** **True Mean #Hops = 3.0233** **True Hop-Range = 0 - 9**	0.5492	3.4833	0 - 9	0.0727
	RMSE	**Estimated Mean #Hops**	**Estimated Hop-Range**	**Time for 1 simulation (sec)**
h = 0.01, *T = 100* **True Mean #Hops = 0.9400** **True Hop-Range = 0 - 5**	0.1841	0.9533	0 - 5	0.0695
h = 0.01, *T = 200* **True Mean #Hops = 2.0667** **True Hop-Range = 0 – 7**	0.3061	2.5467	0 - 11	0.1700
h = 0.01, *T = 300* **True Mean #Hops = 2.9600** **True Hop-Range = 0 – 8**	0.3400	3.8800	0 - 12	0.2166

As the results above show, there is a significant increase of the RMSE when the hopping probability increases. That was expected because a higher hopping probability indicates more hops, true and false, and it gets harder for the edge detector to detect all the true hops and distinguish them from the false ones. When we increase the number of temporal samples, though the hopping probability remains constant, the overall number of frequency hops of the observed signal still increases. Hence, the RMSE increases as well, just not as much as when the hopping probability grows. Moreover, the range of the estimated edges remains close to the real one as the hopping probability increases, which is a good thing. But it is significantly affected by the increase of the number of samples.

The execution time of a single run is not affected as the hopping probability increases, as the above simulations indicate. That is probably because, even for a lower hopping probability, many more than the original hops are detected and considered to be true ones until they are rejected. But, if we increase the number of temporal samples the execution time increases significantly compared to the case where T = 100. That was expected as it takes longer for all the three main steps of the algorithm to perform the same process operations, for example, 200 or 300 samples instead of 100. Is should take approximately twice or three times as long, correspondingly, and as it appears, it does.

Next, we present a few plots showing the algorithm's results for a higher hopping probability and for an increased number of temporal samples.

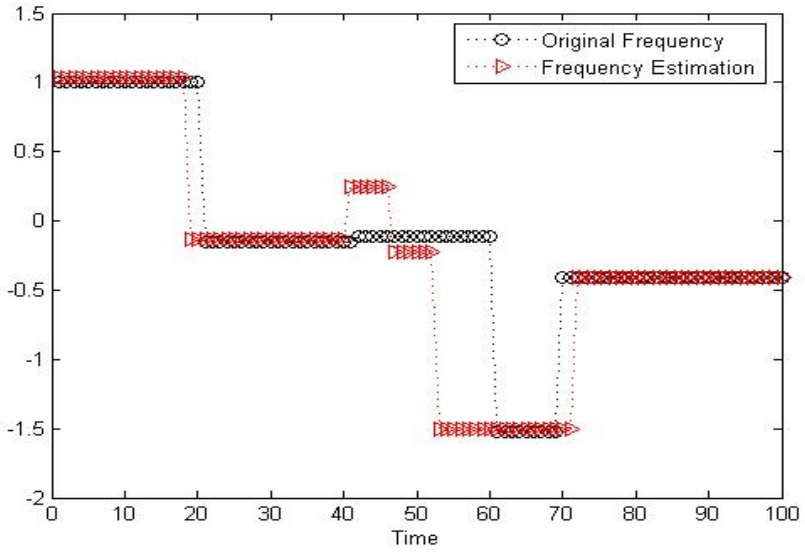


Figure 4: *Typical sample simulation run of the proposed algorithm*

Parameters: $\sigma_n^2=0.2$, $\sigma_A^2=1$, $h=0.02$, $T=100$

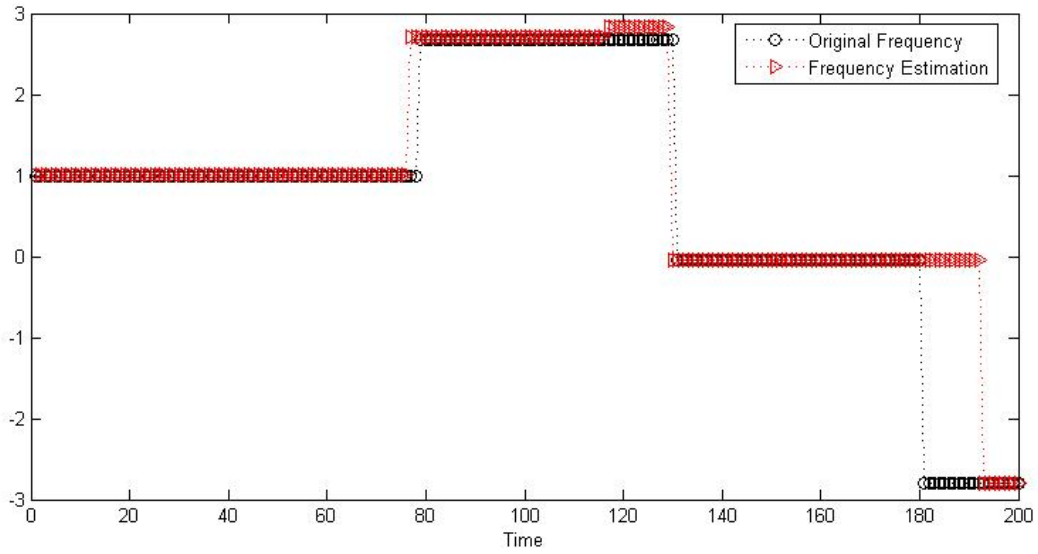


Figure 5: *Typical sample simulation run of the proposed algorithm*

Parameters: $\sigma_n^2=0.2$, $\sigma_A^2=1$, $h=0.01$, $T=200$

3.2 Particle Filtering Algorithm – Simulations

3.2.1 Particle Filtering basics

Sequential state filtering methods are of vital importance to the statistical analysis of stochastic non-linear and/or non-Gaussian state space models. These methods have been developed using the Kalman filter, analytical approximations and particle filtering. In particle filtering (PF), continuous distributions are approximated by discrete random measures which consist of "particles" and associated weights:

$$p(x) = \sum_{n=1}^{N} w_n \delta(x - x_n),$$

where w_n denotes the weights and $\delta(\cdot)$ denotes the Dirac delta functional.

In an on-line (PF) algorithm the aim is to estimate the state at time k using the measurements up to and including time k. The density that captures all information in

these measurements is the posterior density $p(x_k \mid \{y_l\}_{l=1}^{k})$. Direct sampling from the posterior function, though, is not feasible. So, an alternative function which is similar to the posterior and it is called *importance function* is used instead. The basic operation of PF consists of a random initialization of the state distribution and, as new measurements are received, the derivation of new random measures through importance sampling.

In particle filtering, though, the weights have the tendency to become negligible after a few iterations (degeneracy). This drawback can be mitigated via re-sampling techniques.

In this thesis, we refer to the three PF algorithms that were developed by N.D. Sidiropoulos, A. Swami and A. Valyrakis [6] for tracking the frequency of a frequency hopping complex sinusoid. As it has already been mentioned in section 1.2, the data model that was proposed in [6] is actually the one that we used.

The first algorithm they developed uses the prior importance function: $x_{n,k} = f(x_{n,k-1}, u_k)$ and the updated weights: $w_{n,k} = w_{n,k-1} p(y_k \mid x_{n,k})$.The second one uses the optimal importance function:

$$p(x_k \mid x_{n,k-1}, y_k) = \frac{p(y_k \mid x_k) p(x_k \mid x_{n,k-1})}{\int_x p(y_k \mid x) p(x \mid x_{n,k-1}) dx}$$

and the updated weights: $w_{n,k} \propto w_{n,k-1} p(y_k \mid x_{n,k-1}) = w_{n,k-1} D(y_k, x_{n,k-1})$ where $D(y_k, x_{n,k-1}) = \int_x p(y_k \mid x) p(x \mid x_{n,k-1}) dx$. The optimal importance function takes into consideration, contrary to the prior importance function, even the newest available measurement for the particle update. The last algorithm uses a rejection-based sampling of the importance function. For further information regarding the above PF algorithms, please refer to [6].

3.2.2 Particle Filtering simulations

For the simulations that follow we used the Matlab code that was developed by A. Valyrakis [6] for the PF algorithm that uses the optimal importance function. We chose to present this one over the one that uses the prior importance function due to the fact that it results to a better RMSE for a smaller number of particles. And we chose it over the rejection-based sampling of the importance function because its execution time is lesser for a number of particles greater than 1000. We used the multinomial resampling

function for the simulations and the following parameters: h=0.01, MC = 100, $\sigma_A^2 = 1$, T = 100. The results that we obtained were close to those of A. Valyrakis [6].

	RMSE	**Time for 1 simulation (sec)**
σ_n^2=0.2, N = 3000	0.4724	32.6125
σ_n^2=0.2, N = 4000	0.4269	50.8527
	RMSE	**Time for 1 simulation (sec)**
σ_n^2=0.3, N = 3000	0.5194	31.2961
σ_n^2=0.3, N = 4000	0.4852	49.3421
	RMSE	**Time for 1 simulation (sec)**
σ_n^2=0.5, N = 3000	0.7062	32.6783
σ_n^2=0.5, N = 4000	0.6854	47.9872
	RMSE	**Time for 1 simulation (sec)**
σ_n^2=0.8, N = 3000	0.7465	31.9747
σn2=0.8, N = 4000	0.7277	51.4547

The above simulation runs for 3000 and for 4000 particles are for the same data sequences. As we can see, the RMSE decreases as the number of particles is increased. The decrease, though, is getting smaller as the noise power increases. But, it has the advantage that by increasing the number of particles we can decrease the RMSE. Notice that the complexity of particle filtering is O(NT), which is manageable and therefore, we can increase the number of particles until we reach a satisfying RMSE. However, though, the constants are large.

	RMSE	Time for 1 simulation (sec)
h = 0.01, T = 100, N = 4000	0.4269	50.8527
h = 0.02, T = 100, N = 4000	0.7543	67.6559
h = 0.03, T = 100, N = 4000	0.8814	87.3280

	RMSE	Time for 1 simulation (sec)
h = 0.01, T = 100, N = 4000	0.4269	50.8527
h = 0.01, T = 200, N = 4000	0.5677	121.7219
h = 0.01, T = 300, N = 4000	0.7919	215.8426

The simulations show that the algorithm is affected by both the increase of the hopping probability and the increase of the number of samples. The effect is greater for the hopping probability's value change as the RMSE is mostly caused by the difficulty of the hop timing estimation. But, the execution time for the increase in samples is affected way more than for the increment of the hopping probability. The execution time of the algorithm in general is less than O(NT) and it also depends on the hopping probability.

Chapter 4

The final chapter contains the simulation results, for the exact same data, of the RWS approach in addition to the two previously presented algorithms. Later on in the chapter, we compare the proposed algorithm with the other two approaches.

4.1 Comparison – Conclusions

The simulations results that follow are for the same data sequences for all three algorithms: spectrogram segmentation - based, particle filtering and RWS. The parameters we used are: MC = 100, T = 100, σ_A^2=1 and h = 0.01, unless it is indicated otherwise inside the table.

RMSE	**Proposed Algorithm**	**Particle Filtering Algorithm (N = 4000)**	**RWS Estimation**
σ_n^2=0.2	0.1709	0.4003	0.6525
σ_n^2=0.3	0.2588	0.4852	0.6482
σ_n^2=0.5	0.2609	0.5165	0.7922
σ_n^2=0.8	0.3827	0.7277	0.8374
RMSE	**Proposed Algorithm**	**Particle Filtering Algorithm (N = 4000)**	**RWS Estimation**
σ_n^2=0.2, *h = 0.02*, T = 100	0.4802	0.7543	1.1262
σ_n^2=0.2, *h = 0.03*, T = 100	0.6268	0.8814	1.4281
RMSE	**Proposed Algorithm**	**Particle Filtering Algorithm (N = 4000)**	**RWS Estimation**
σ_n^2=0.2, h = 0.01, *T = 200*	0.2867	0.5677	1.0552
σ_n^2=0.2, h = 0.01, *T = 300*	0.3584	0.7919	1.3283

Time for 1 simulation (sec)	**Proposed Algorithm**	**Particle Filtering Algorithm (N = 4000)**	**RWS Estimation**
σ_n^2=0.2	0.0748	53.7275	0.0545
σ_n^2=0.3	0.0769	49.3421	0.0651
σ_n^2=0.5	0.0791	45.4673	0.0633
σ_n^2=0.8	0.0983	31.9747	0.0687
Time for 1 simulation (sec)	**Proposed Algorithm**	**Particle Filtering Algorithm (N = 4000)**	**RWS Estimation**
σ_n^2=0.2, *h = 0.02*, T = 100	0.0948	67.6559	0.0762
σ_n^2=0.2, *h = 0.03*, T = 100	0.0967	87.3280	0.0658
Time for 1 simulation (sec)	**Proposed Algorithm**	**Particle Filtering Algorithm (N = 4000)**	**RWS Estimation**
σ_n^2=0.2, h = 0.01, *T = 200*	0.1305	121.7219	0.1102
σ_n^2=0.2, h = 0.01, *T = 300*	0.1617	215.8426	0.1631

Spectrogram segmentation-based algorithm vs. RWS estimation

According to the simulation runs above, both algorithms are just as fast. Notice that the RWS is in fact the first step of the proposed algorithm. The latter performs better in terms of RMSE than the RWS for all the different standard deviations of white Gaussian noise. That proves that the idea we had (section 1.4, chapter 1) about using edge detection and how that would help decrease the frequency estimation error, was insightful.

The RWSE, although it does not offer a better RMSE, is a bit more resistant to noise as the only significant RMSE-increment occurs for σ_n^2=0.5 unlike the proposed algorithm's RMSE that rises significantly twice: σ_n^2=0.3 and σ_n^2=0.8.

Now, as the hopping probability rises or the quantity of temporal samples increases, both algorithms show significant weakness in terms of RMSE. But, relatively speaking, the proposed algorithm acts more efficiently: its RMSE increment each time is less than half the corresponding one for the RWS estimation.

Spectrogram segmentation-based algorithm vs. PF algorithm

As indicated by the simulation runs, both algorithms show a significant increase in RMSE for $\sigma_n^2=0.3$, the RMSE for up to $\sigma_n^2=0.5$ is pretty much the same and $\sigma_n^2=0.8$ the RMSE increment is great. Additionally, they are both sensitive to the increment of the hopping probability and the quantity of samples (less to the latter). In other words, they provide a much worse RMSE in those cases. But, the proposed algorithm, compared to the PF estimation for N=4000, performs much better in terms of RMSE for all the scenarios above. PF though has the advantage that by increasing the number of particles, if we can afford to pay the complexity cost, we can decrease the frequency estimation error. If there is no saturation after a certain number of particles, than the PF algorithm could perhaps compensate this great difference in RMSE (0.25–0.35) with the spectrogram segmentation-based algorithm.

The time it takes the PF algorithm to process a data sequence for 4000 particles is too long in comparison with the corresponding one of the proposed algorithm. This execution time will keep growing considerably if we keep increasing the number of particles, while the time the proposed algorithm needs remains constant.

A major drawback of the presented algorithm over the RWS and the PF method is that it is non-causal: it uses future samples to go back and detect previous hops and then estimate the frequency at a "past" and at a "present" time. Although we could use a causal RWS (see section 2.2) the method does not apply for a causal approach. That is because even if we could use dynamic edge detection, in the final step the periodogram would still use future samples for the frequency estimation.

The plots that follow show the performance in terms of the RMSE of all three algorithms plus a few sample runs for a variety of white Gaussian noise as well as for a higher hopping probability and for a number of temporal samples more than T = 100.

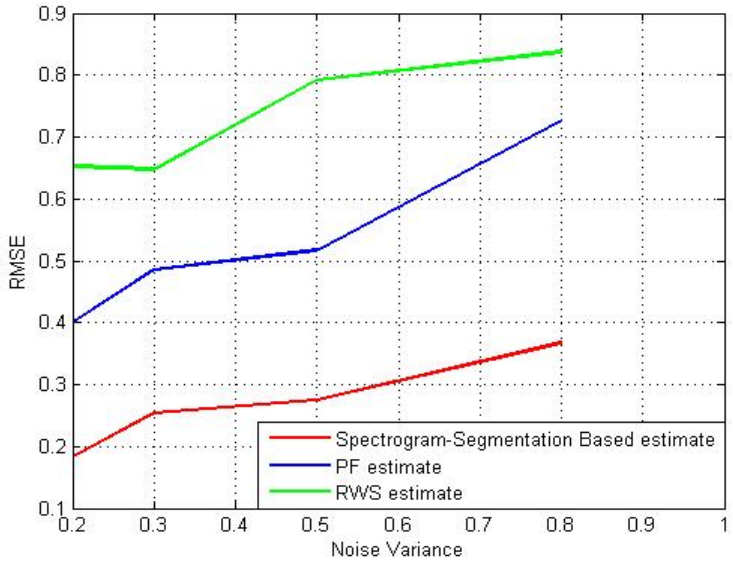


Figure 6: *Performance of the algorithms in terms of the RMSE*

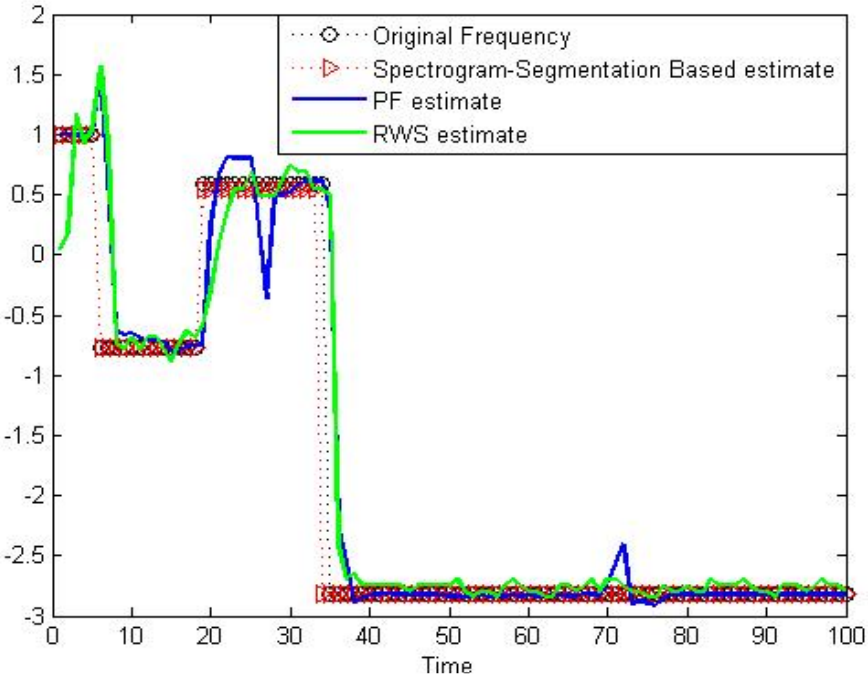


Figure 7: *Typical sample simulation run of all three approaches*

Parameters: $\sigma_n^2=0.2$, $\sigma_A^2=1$, $h=0.01$, $T=100$

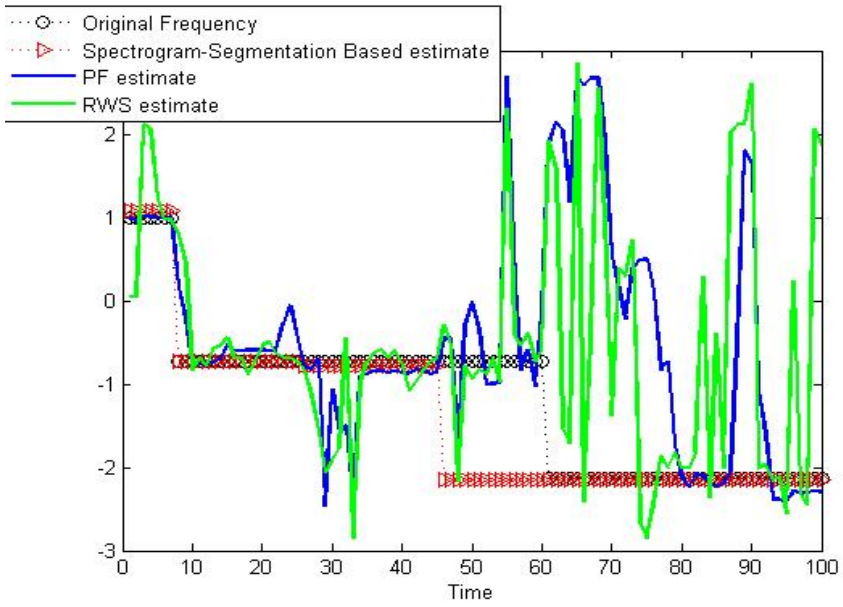


Figure 8: *Typical sample simulation run of all three approaches*

Parameters: $\sigma_n^2=0.5$, $\sigma_A^2=1$, $h=0.01$, $T=100$

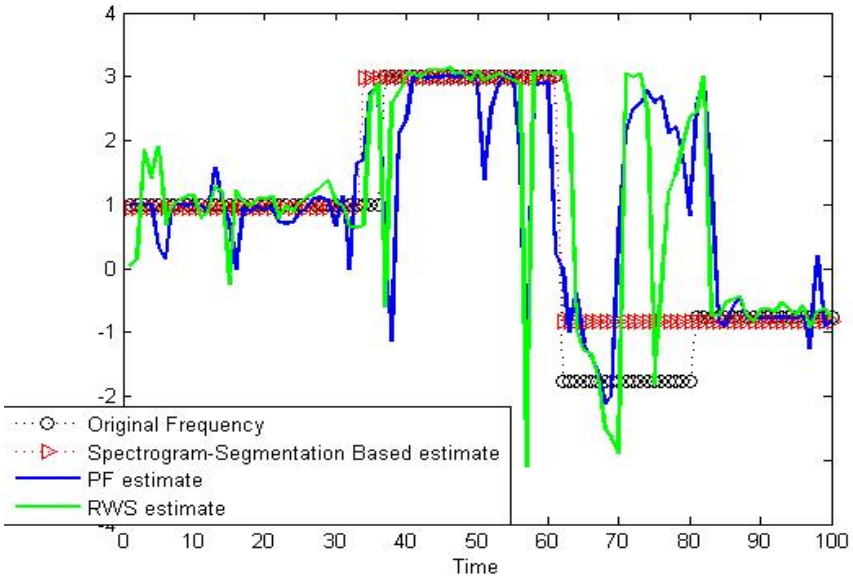


Figure 9: *Typical sample simulation run of all three approaches*

Parameters: $\sigma_n^2=0.2$, $\sigma_A^2=1$, $h=0.02$, $T=100$

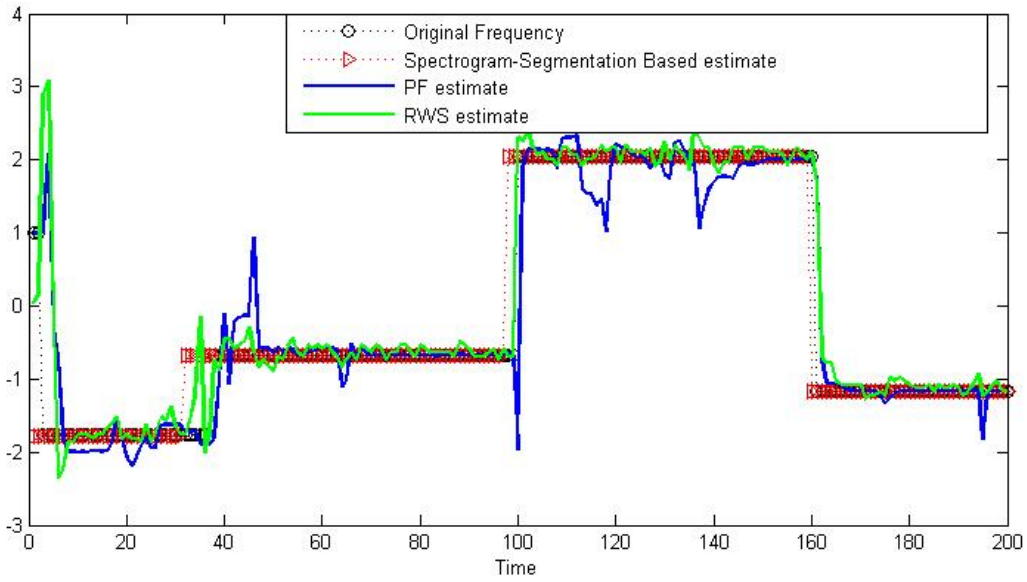


Figure 10: *Typical sample simulation run of all three approaches*

Parameters: $\sigma_n^2=0.2$, $\sigma_A^2=1$, $h=0.01$, $T=200$

Conclusions

The spectrogram segmentation-based algorithm we have developed makes use of the RWS estimation in a more sophisticated and intuitive way. As a result, we got a more efficient algorithm with no complexity cost whatsoever.

Now, the PF algorithm can compete the performance of the proposed algorithm only by increasing the number of particles and by paying great, relatively speaking, complexity cost. That of course, if there is no saturation after a particular number of particles. Even so, at least for the occasion where the temporal samples are 100 and the standard deviation of white Gaussian noise is $\sigma_n^2=0.2$, the spectrogram segmentation-based algorithm is more attractive due to its good estimation (RMSE 0.15-0.2) and its really small complexity cost. Still, though, the presented algorithm is at disadvantage in that it is non-causal: it uses future samples for the hop-detection and the frequency estimation.

Bibliography

[1] Barbarossa, S. and Scaglione, A., "Parameter estimation of spread spectrum frequency-hopping signals using time-frequency distributions", Signal Processing Advances in Wireless Communications, 1997 First IEEE Signal Processing Workshop on Volume , Issue , 16-18 Apr 1997 Page(s):213 – 216

[2] Andrieu, C., Doucet, A., Tadic, V.B., "On-Line Parameter Estimation in General State-Space Models", Decision and Control, 2005 and 2005 European Control Conference. 44th IEEE Conference on Volume, Issue, 12-15 Dec. 2005 Page(s): 332 - 337

[3] Petre Stoica, Randolph Moses, "Introduction to spectral analysis". Printed: February 1997, Publisher: Prentice Hall

[4] Chapter 1: "Advanced edge detection techniques" of "Algorithms for image processing and computer vision" by James R. Parker. Printed: November 1996, Publisher: Wiley, John & Sons, Incorporated

[5] Canny, J.F., "A computational approach to edge detection", IEEE Trans Pattern Analysis and Machine Intelligence, 8(6):679-698, Nov 1986

[6] Sidiropoulos, N.D. , Swami, A. and Valyrakis, A., "Tracking a Frequency Hopped Signal Using Particle Filtering", Acoustics, Speech and Signal Processing, ICASSP 2006 Proceedings, 2006 IEEE International Conference on 14 - 19 May 2006, Volume: 3, On page(s): III-III.

[7] Master Thesis by J.F. Canny, "Finding Edges and Lines in images", MIT Library, Artificial Intelligence Laboratory