8-8-2007

Frequency Estimation Using Time-Frequency Based Methods

Cuong Mai
University of New Orleans

Frequency Estimation Using Time-Frequency Based Methods

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Engineering
Electrical Engineering

by

Cuong Mai

B.S. University of New Orleans, 2003

August 2007

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Any periodic signal can be decomposed into a sum of oscillating functions. Traditionally, cosine and sine segments have been used to represent a single period of the periodic signal (Fourier Series). In more general cases, each of these functions can be represented by a set of spectral parameters such as its amplitude, frequency, phase, and the variability of its instantaneous spectral components. The accuracy of these parameters depends on several processing variables such as resolution, noise level, and bias of the algorithm used. This thesis presents some background of existing frequency estimation techniques and proposes a new technique for estimating the instantaneous frequency of signals using short sinusoid-like basis functions. Furthermore, it also shows that the proposed algorithm can be implemented in a popular embedded DSP-microprocessor for practical use. This algorithm can also be implemented using more complex features on more resourceful processing processors in order to improve estimation accuracy.

Keywords: *frequency estimation, oscillating functions, Digital Signal Processing, instantaneous spectral components, instantaneous frequency, periodic signal*

INTRODUCTION

Instantaneous frequency estimation is a signal processing component which is vital for many application fields, including analysis of radar signals, seismic reflection, and electromagnetic interference [1]-[4]. Instantaneous frequency is defined as the derivative of the argument $f(t)$ of a signal having the form $cos(f(t))$, namely $\frac{df(t)}{dt}$. Precise interpretation of complex signals that consist of various time-frequency structures requires decomposing the signals into parametric, redundant, and well-localized components in the time-frequency plane. These localized functions are often called time-frequency atoms. Decomposing signals in time-frequency components gives important information about the signal, when such information cannot be obtained by using the classical time-domain or frequency-domain analysis methods [5]. For instance, rotation and shearing [7] characteristics of frequency-varying signals can be obtained in the time-frequency plane, as it will be discussed in detail in Chapter 1.

This thesis presents an overview of instantaneous frequency estimation methods, including those based on Short Time Fourier Transform (STFT), Wavelet Transform, and Chirplet Transform. Then, two practical methods of Chirplet Transform-based frequency estimation for multi-component signals are discussed. A new method is proposed, and its hardware implementation on a TI6713 board is discussed. The limitations and strengths of the hardware used are also discussed. Moreover, a MATLAB implementation is presented with the purpose of expanding the proposed method into more powerful computing processors. The thesis concludes with an analysis of the results obtained from both hardware and MATLAB implementations.

CHAPTER 1

CURRENT METHODS FOR ANALYZING INSTANTANEOUS FREQUENCY

I. Short Time Fourier Transform (STFT) and Wavelet Transform

In the recent past, researchers became aware of the limitations of frequency-domain methods. The Fourier Transform (FT), which was traditionally used for frequency analysis, was able to yield perfect reconstruction of signals; however, it was not able to provide a meaningful interpretation of the signal's frequency content in the case where the signal varies with time. Later, researchers focused on time-localized methods which allowed them to observe how the signal's spectrum evolves over time by performing frequency analysis within short time signal segments. These methods are called time-frequency (TF) methods. One of these methods is the short-time Fourier transform (STFT). This method involves computing the FT of a signal $s(t)$ using a window of short duration. The process is repeated for different shifts of the window, thus obtaining the spectrum of the signal at different time intervals. A multiple-scale version of this approach involved stretching the window size gradually, in order to compute the STFTs for each window size. As shown in

Figure 1, stacking these STFT's on top of one another results in a continuous volumetric representation of the signal $s(t)$, which is a function of time, frequency, and window size [6].

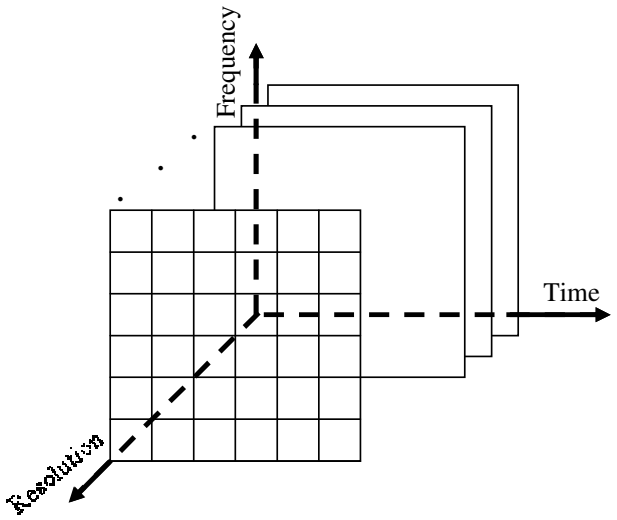

Figure 1 : 3-D parameter space based on the use of multiple windows

Another time-frequency representation is the Wavelet Transform which involves computing the inner product of the signal $s(t)$ under analysis with a family of translates and dilates of one basic primitive called the mother wavelet. A member of the wavelet family is produced by a particular 1-D affine coordinate transformation acting on the time axis of the mother wavelet. This geometric transformation is parameterized by two numbers corresponding to the amounts of translation and dilation. By taking the inner products of the signal $s(t)$ with many members of the two-parameter wavelet family, the continuous wavelet transform is formed [6].

II. Chirplet Transform

A different approach for obtaining a time-frequency representation of $s(t)$ is based on the decomposition of $s(t)$ using basis functions. This approach is based on an assumption that, in general, a signal can be expressed as a weighted sum of mono- component signals. In the case of the FT representation, these mono-component signals are single frequency components represented by $A\,e^{j\omega t}$, where ω is the component's frequency, and A is its amplitude. The Chirplet Transform approach uses a different representation of the signal components, which in this case are called chirplets. It will be discussed in detail in section II.1 that chirplets are short-timed signals that depend on several parameters, in contrast to the single-frequency components, $A\,e^{j\omega t}$, used in FT, and to the wavelets used in Wavelet Transforms.

The concept of Wavelet Transform and Fourier Transform processing is similar to that of Chirplet Transform processing, in the sense that the original signal is projected on a set of basis functions. The Chirplet Transform, as formulated by Mann and Haykin [6] is "a multi-dimensional parameter space whose coordinate axes correspond to the pure parameters of planar affine coordinate transformations in the time-frequency plane." Compared to other linear Time-Frequency representations, such as STFT or the Wavelet Transform, the Chirplet Transform represents 1-D time-domain signals by multi-dimensional functions. Most often, the function is five-dimensional, where the five dimensions are:

- amplitude scaling,
- chirping in time,
- chirping in frequency,
- shearing in time,
- shearing in frequency.

The mapping of a signal in the five-dimensional space, whose dimensions are defined above, helps to overcome the resolution problem from which other time-frequency representations suffer [7].

II. 1. Gaussian Chirplet

A simple chirplet can be derived by applying simple mathematical operations to a Gaussian enveloped function. The function can be thought of as the primitive that generates a family of chirplets. The primitive function is defined as follows:

$$g'_{t_c,f_c,\sigma}(t) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(1/2)(t-t_c/\sigma)^2} e^{j2\pi f_c(t-t_c)+j\phi} \qquad (1)$$

where $j=\sqrt{-1}, t_c \in \Re$ is the center of the energy concentration in time; $f_c \in \mathbb{R}$ is the center frequency, $\sigma \in \mathbb{R} > 0$ is the spread of the Gaussian envelope, and $\phi \in \mathbb{R}$ is called the phase shift. The three subscripts of function *g'* represent the three chirplet's degrees of freedom. If the primitive function is required to have unit energy, it needs to be raised to ½ and multiplied by an appropriate normalization constant. The phase parameter φ does not appear in (2) for the purpose of simplifying the expression.

$$g'_{t_c,f_c,\sigma}(t) = \sqrt{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{t-t_c}{\sigma}\right)^2}} e^{j2\pi f_c(t-t_c)} = \frac{1}{\sqrt{\sqrt{\pi}\Delta_t}} e^{-\frac{1}{2}\left(\frac{t-t_c}{\Delta_t}\right)^2} e^{[j2\pi f_c(t-t_c)]} \quad (2)$$

where $\Delta_t = \sqrt{2}\sigma$

This primitive function can also be thought of as the product of an envelope and the harmonic oscillation component. A family of Gaussian chirplets is formed by retaining the envelope of the primitive and replacing the harmonic oscillation with a linear frequency modulated chirp. For example:

$$g_{t_c,f_c,\log(\Delta_t),c}(t) = \frac{1}{\sqrt{\sqrt{\pi}\Delta_t}} e^{-\frac{1}{2}\left(\frac{t-t_c}{\Delta_t}\right)^2} e^{j2\pi[c(t-t_c)^2 + f_c(t-t_c)]} \quad (3)$$

The expression above defines a Gaussian chirplet with four parameters: time center t_c, frequency center f_c, log-duration log (Δt), and chirprate c [6].

A chirp and a chirplet in time are shown in Figure 2.

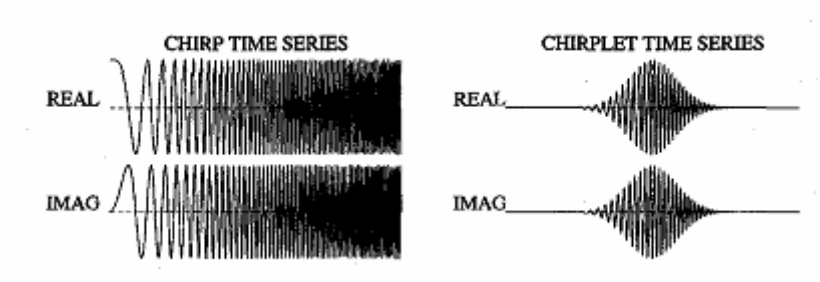

Figure 2: Chirp and Chirplet in time (Real and Imaginary components) (Image by Mann and Haykin) [6]

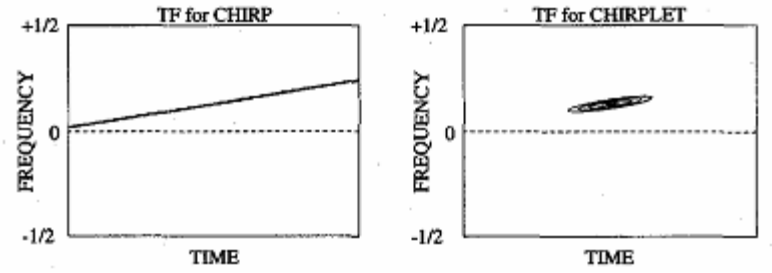

Figure 3: Time-Frequency scale of Chirp and Chirplet (Image by Mann and Haykin) [6]

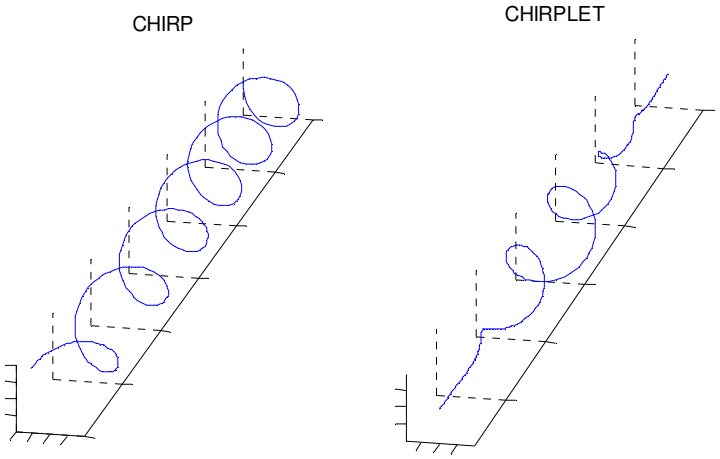

Figure 4: 3-dimensional helix view of Chirp and Chirplet in time

Figure 3 shows the chirp and chirplet in the time-frequency scale. The chirp is characterized by a linear increase in frequency. Figure 4 shows a 3-D helix view of the chirp and chirplet as they change in time. As shown in Figure 3 (right), a chirplet is represented as ***a cell of ellipsoidal shape***. The chirplet also has the same linear increase in frequency but with varying amplitude (growing first and decaying later) [6].

II.2. Chirplet Operators

The Chirplet Transform is capable of rotating each cell of the time-frequency plane similar to the one shown in Figure 4, as well as shearing it along the time and frequency axes. The Chirplet Transform's capability of shaping the time-frequency cells presents new opportunities of optimizing the time-frequency resolution according to the analyzed signal. For example, it is possible to rotate and shear each cell according to the local slope of the trajectory of the analyzed instantaneous frequency, thus giving the opportunity of better tracking the frequency's evolution versus time. These degrees of freedom are provided by modifying the chirp's parameters in order to perform basic chirp operations such as the ones described next:

a. Chirping in time

Chirping in time is obtained by multiplying the mother chirplet (Figure 5), or the primitive function, $g(t)$ with a linear Frequency Modulated (FM) signal, chirp ($e^{j2\pi\frac{c}{2}t^2}$) where c is the chirp-rate. It causes a rotation of each cell as well as its shear along the frequency axis. The rotation angle with respect to the time axis is α which depends on the chirp-rate c. The slope of the cell on the time-frequency plane is equal to the value of c (Figure 6) [7].

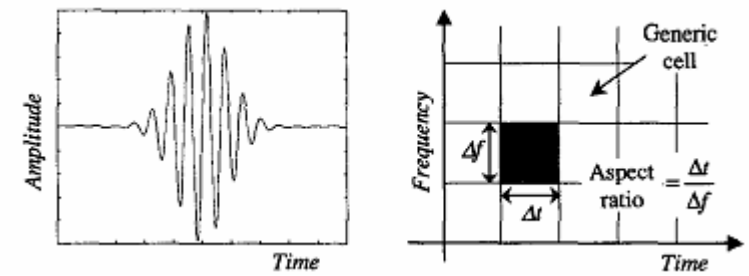

Figure 5: An original chirp function (Image by Angrisani and D'Arco) [7]

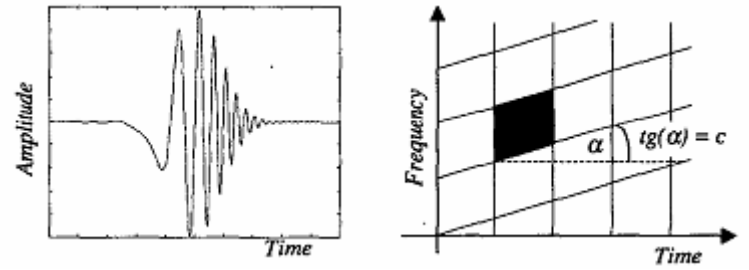

Figure 6: Chirping in Time (Image by Angrisani and D'Arco) [7]

b. Chirping in frequency (time shearing)

Chirping in frequency (time shearing) [6] is given by convolving, in the time domain, the mother chirplet *g(t)* with another FM linear chirp $(-jd)^{-\frac{1}{2}} e^{j2\pi\frac{1}{2d}t^2}$ where the parameter *d* accounts for the shear amount along the time axis imposed on the cell. In the frequency domain, this is accomplished by multiplying the Fourier Transform of the mother chirplet, *G*(*t*), and the function $e^{-j2\pi\frac{d}{2}f^2}$ which can be considered a chirp in the frequency domain with the chirp-rate equal to *d* [7]. The rotational angle with respect to the frequency axis is β which depends on the chirp-rate *d*. As with chirping in time, the slope of the cell with respect to the frequency axis is equal to *d.*

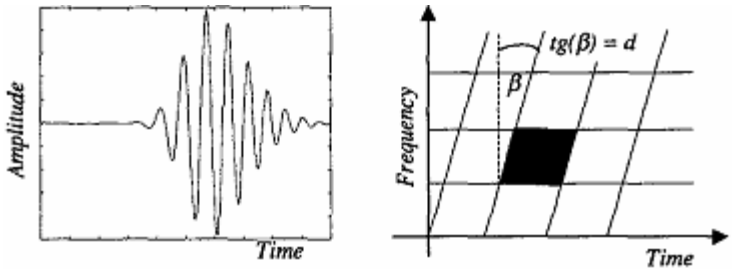

Figure 7: Chirping in Frequency (Image by Angrisani and D'Arco) [7]

c. Time translation

Time translation is done by shifting the mother chirplet $g(t)$ by t_c in time where t_c is the time offset [7].

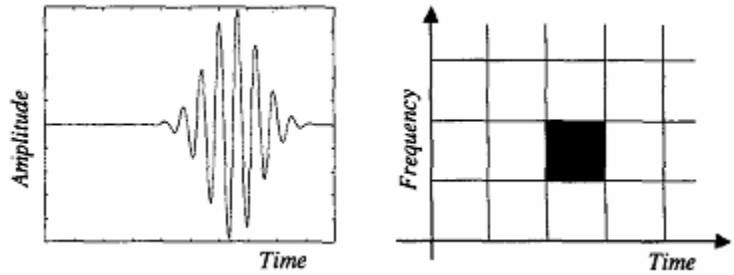

Figure 8: Time translation $g(t - t_c)$ (Image by Angrisani and D'Arco) [7]

d. Frequency-translation:

Frequency translation is done by multiply the mother chirplet $g(t)$ by $e^{j2\pi fct}$ or $g(t)e^{j2\pi fct}$ where f_c is the amount of shift in frequency [7].

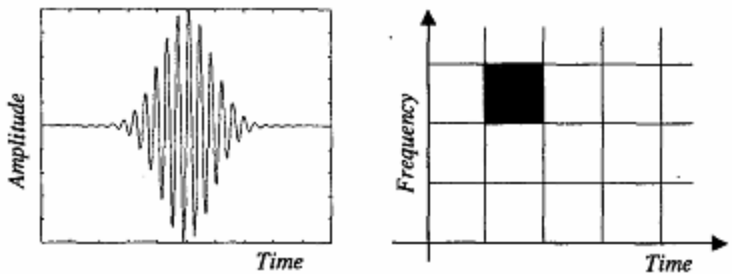

Figure 9: Frequency Translation (Image by Angrisani and D'Arco) [7]

e. Dilation in Frequency/ Contraction in Time:

Changing the sampling rate or resolution of the signal in time cause the cell to be either dilated or contracted in time. When that happens, the reverse effect is caused in the frequency scale [7].

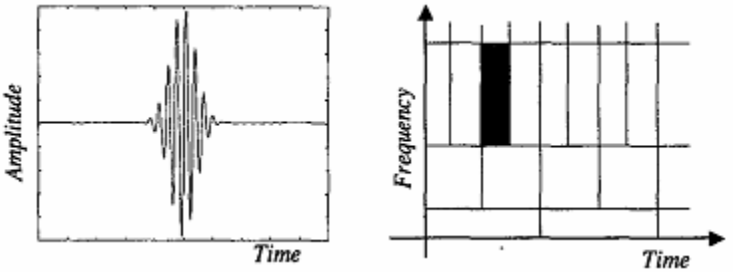

Figure 10: Frequency dilation/ time contraction represented by $\frac{1}{\sqrt{|\Delta_t|}} g\left(\frac{1}{\Delta_t}\right)$ (Image by Angrisani and D'Arco) [7]

All of the above operations can be combined to a complete analytic expression of the Chirplet Transform of a signal $s(t)$:

$$CT_s(t_c, f_c, a, c, d) = \int s(t_c) \times h(t - t_c, f_c, a, c, d) dt \qquad (4)$$

with the kernel $h(\tau)$ given by:

$$h(t-t_c, f_c, a, c, d) = \left(\frac{1}{\sqrt{-jda}}\right)\int g\left[\frac{(t-t_c)}{a} - v\right] e^{j\pi\left\{2f_c t_c + c\left[\frac{(t-t_c)}{a} - v\right]^2 + \frac{v^2}{d}\right\}} dv \quad (5)$$

t : time

f: frequency

a: scaling parameter

The full continuous chirplet transform (CCT) can also be written using a different notation as:

$$S_{t_c, f_c, \log(\Delta_t), c, d} = \left\langle C_{t_c, f_c, \log(\Delta_t), c, d}\, g(t) \mid s(t) \right\rangle \quad (6)$$

where t_c, f_c, $log(\Delta_t)$, c, d are the time shift, frequency shift, resolution (based on window size), chirp-rate in time, and chirp-rate in frequency.

For any operator in the time domain, there is an equivalent operator in the frequency domain or in the time-frequency plane or in any other reasonable coordinate space in which one wants to work in. When using more than one operator, it should be cautioned that operators do not commute, including the frequency-shift and time-shift operators. For example, if time-shift is applied first and then frequency shift is applied next. This is not equivalent to applying a frequency shift followed by a time shift, since the result would contain a different phase shift [6].

III. CHIRPLET TRANSFORM AND FREQUENCY ESTIMATION

The Chirplet Transform projects the input signal onto a set of functions that are obtained by modifying an original function $g(t)$, or the mother chirplet as described in Part II. The most popular chirp operators being used are time shifting, frequency shifting, scaling, chirping in time, and chirping in frequency [7]. Because of the variety of operators being used, the Chirplet Transform reveals more information about the original signal $s(t)$ compared to the traditional FT. However, Chirplet Transform can be very time-consuming since each chirplet parameter's range must be large enough to obtain a large number of chirplets.

IV. MODIFIED CHIRPLET TRANSFORM FOR INSTANTANEOUS FREQUENCY ESTIMATION

IV.1. Mathematical derivation

Angrisani, L. and D'Arco, M. proposed a method of modified Chirplet Transform to improve the shape and aspect ratio of the time-frequency cell, to better match the local features of the analyzed signal [7]. To accomplish this, a modification of the mother chirplet, $g(t)$ is introduced by multiplying the chirplet function by the following term:

$$e^{j2\pi\frac{k}{6}t^3} \tag{7}$$

where k is the curvature parameter.

With this modification, each cell in the time-frequency plane can be curved by an amount proportional to the value of the curvature parameter k. The authors claimed that this parameter will

enable the Chirplet Transform to track the evolution versus time of the instantaneous frequency of each analyzed component more closely in the presence of instantaneous frequencies characterized by high dynamics. The authors also claimed that in high dynamic environments, this modified version of Chirplet Transform achieves better resolution than the linear approximation provided by the traditional Chirplet Transform [7].

In particular, the authors proposed using a Gaussian window function as the mother chirplet to achieve the best time-frequency resolution. Its expression is:

$$g(t) = \sqrt{\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{t}{\sigma}\right)^2}} \tag{8}$$

This mother chirplet has unit energy and the parameter σ accounts for its time extent. In Figure 11b, the noticeable bending of the time-frequency cell is visible. This bending effect is claimed by the authors as effectively corresponding to the trajectory points of the analyzed signal (in the time-frequency plane). Moreover, because the time-frequency cell representation of this mother chirplet has elliptical contours on the time-frequency plane, analyzing chirping in frequency is not necessary. On the other hand, the scaling parameter, a, at any point in the time-frequency plane has to establish only the aspect ratio of the cell centered at that point, not the position of the cell along the frequency axis [7].

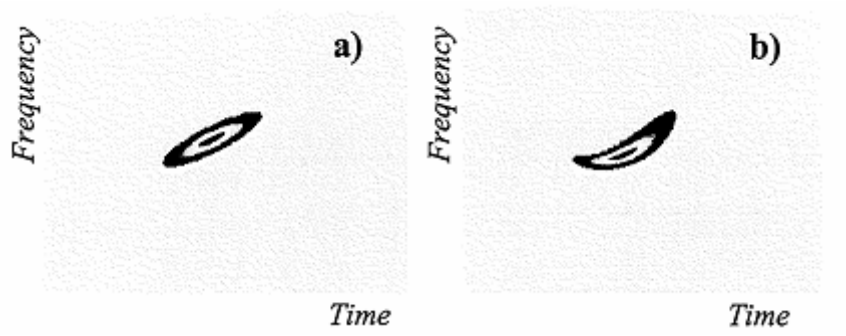

Figure 11: a) Time-Frequency plot from a traditional Chirplet Transform. b) Time-Frequency plot from a modified Chirplet Transform (Image by Angrisani and D'Arco) [7]

The final expression of the kernel, $h\,(t\text{-}t_c, f_c, a, c, d)$, of the modified Chirplet Transform is:

$$h(t-t_c, f_c, a, c, k) = \sqrt{\frac{1}{\sqrt{2\pi}a\sigma} e^{-\frac{1}{2}\left(\frac{t-t_c}{a\sigma}\right)^2}} \times e^{j\pi\left[f_c(t-t_c)+c(t-t_c)^2+\frac{k}{3}(t-t_c)^3\right]} \qquad (9)$$

2. Computer Algorithm

a. Chirp-rate Establishment

Initially, a traditional Chirplet Transform with no chirping in time ($c=0$), no bending effect ($k=0$), and unitary scale parameter ($a = 1$) is evaluated for each Δt of the test signal. The results give a rough estimate of how many components the signal consist of. A peak location algorithm [8] is applied to the same results to obtain a coarse time evolution of the instantaneous frequency of each component. Then, the slope at each point (t^*, f^*) on the time-frequency plane is evaluated. This slope is calculated by using the incremental ratio, r:

$$r = \frac{f_2 - f_1}{t_2 - t_1} \tag{10}$$

where (f_2, t_2) and (f_1, t_1) are the coordinates of the two points of the same trajectory that are close to (t^*, f^*) so that $t_1 < t^* < t_2$ and $f_1 < f^* < f_2$.

The value of *r* characterizing a generic point *(t*, f*)* of each reconstructed trajectory is equal to the value of the chirp rate *c* which characterizes the cell centered at the same point in the time-frequency plane [7].

b. Curvature Evaluation

Subsequently, for each time-frequency point at *(t*, f*)*, a Gaussian mother chirplet is established with time and frequency shifted by *t** and *f**, respectively. Chirping in time is also applied with chirp rate *r*. Then, the value of the curvature parameter, *k*, for the same cell is iteratively incorporated into the mother chirplet using (12). Finally, the mother chirplet is projected onto the signal until the results reach a maximum. The value of *k* which results in that maximum will be the desired curvature value of the cell [7].

c. Scaling Adjustment

Lowering the value of the scale parameter, *a*, results in reducing the value of Δt. The values of the parameters *a* and *k* also depend on the sample rate and the number of samples acquired from the

test signal. These properties are used for computing the scaling factor for the cell on the time-frequency plane [7].

d. Instantaneous Frequency Estimation

Frequency estimation is done by using the values of the chirp rate, curvature, and scale obtained previously to characterize all the points in the time-frequency plane belonging to the cell centered at *(t*, f*)*. The process is repeated for the remaining points of the time-frequency plane. To efficiently accomplish this, the authors suggested generating a map of the optimal values of parameters throughout the time-frequency plane. The modified Chirplet Transform of the signal is evaluated by making use of the mapped parameter values to reconstruct the trajectory of each component's instantaneous frequency. An enhanced peak location algorithm is suggested by the authors to perform this reconstruction [7].

3. Testing and Results

a. Test on Mono-component signals

Testing was conducted by the authors on a mono-component signal which has the following expression in the time domain:

s(t) = a(t)x(t) where $0 < t < 102.4\ \mu s$; $a(t) = e^{-\pi\left(\frac{5*10^5 t-28}{15}\right)^2}$

$$x(t) = \cos[2\pi(2.004\cdot 10^6 t - 7.26\cdot 10^{10} t^2 + 1.1882\cdot 10^{15} t^3 - 5.375\cdot 10^{19} t^4)]$$

The test signal is sampled with a sample rate of 5 MSamples/s. 512 samples are acquired. Its plot in the time domain is:

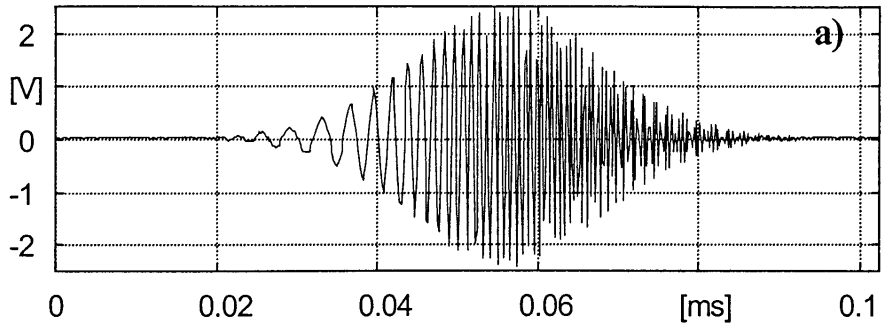

Figure 12: Test signal *s(t)* in the time domain (Plot by Angrisani and D'Arco) [7]

The results acquired in the time-frequency plane using the traditional and modified Chirplet Transform are shown in Figure 13 and Figure 14, respectively. The noticeable bend can be observed clearly in the result from the modified Chirplet Transform. The authors stated that the maximum difference between the test signal and the synthesized signal from the modified Chirplet Transform is about 1% while it is 3% for the traditional Chirplet Transform [7].

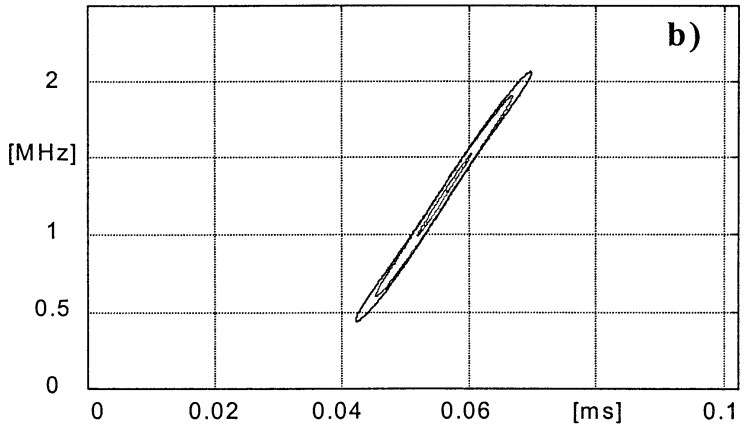

Figure 13: Time-frequency plot of result obtained from the traditional Chirplet Transform (Plot by Angrisani and D'Arco) [7]

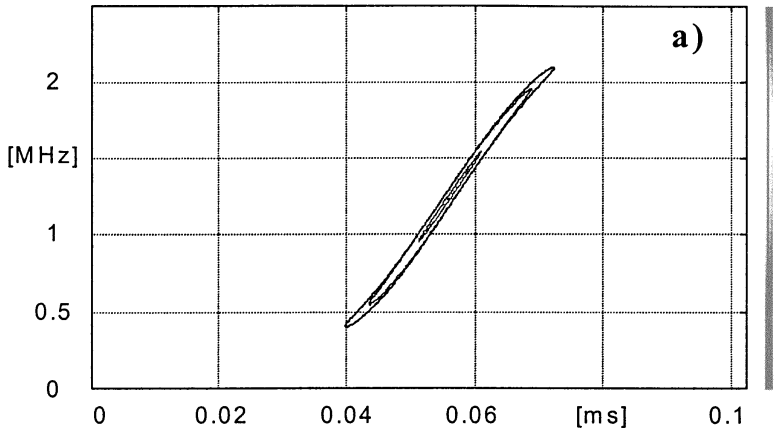

Figure 14: Time-frequency plot of result obtained from the modified Chirplet Transform (Plot by Angrisani and D'Arco) [7]

b. Test on multi-component signal

A test was conducted by the authors on a test signal which has two components [7] with the following expression:

$$s(t) = x(t) + y(t) \qquad 0 < t < 102.4\mu s$$
$$x(t) = \cos[2\pi(1.9041 \cdot 10^6 t - 1.0252 \cdot 10^{10} t^2 + 3.482 \cdot 10^{13} t^3)]$$
$$y(t) = \cos[2\pi(2.0041 \cdot 10^{16} t - 1.0252 \cdot 10^{10} t^2 + 3.482 \cdot 10^{13} t^3)]$$

The same sample rate (5 Msamples/s) and the same number of samples (512 samples) were used for this test. Figure 15 shows the graphical plot of the input signal in time.

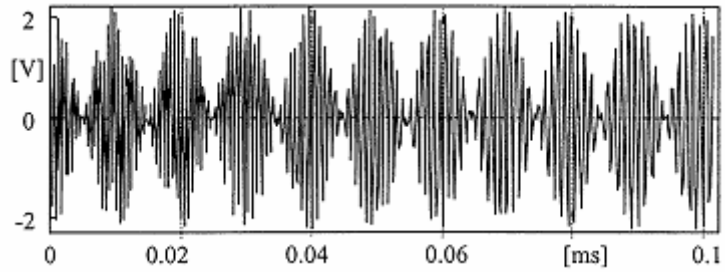

Figure 15: Time plot of two component test signal (Plot by Angrisani and D'Arco) [7]

The result (Figure 16) is then compared with the result from a reassigned STFT on the same test signal in Figure 17. Using these results to reconstruct the trajectories in both cases, the reconstructed trajectories in the case of the modified Chirplet Transform are observed as smooth and continuous for both components while they are observed as abrupt and partially visible when using STFT. As a result, the authors concluded that the modified Chirplet Transform performed better than STFT at separating multiple closely-spaced spectral component signals [7].

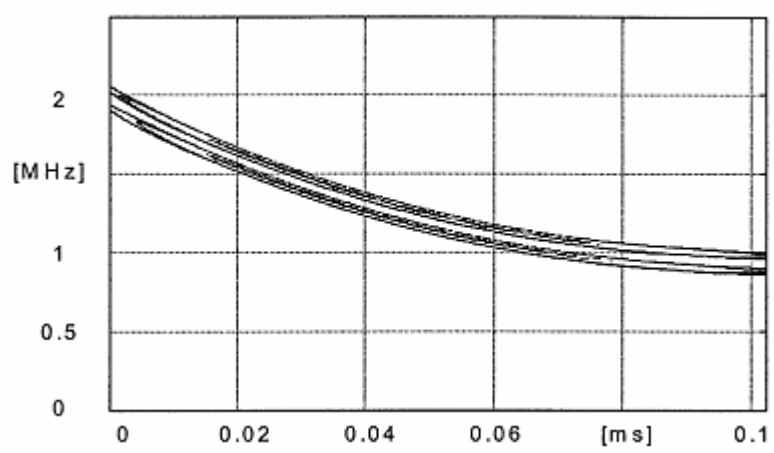

Figure 16: Result obtained from Modified Chirplet Transform on a two component signal (Plot by Angrisani and D'Arco) [7]

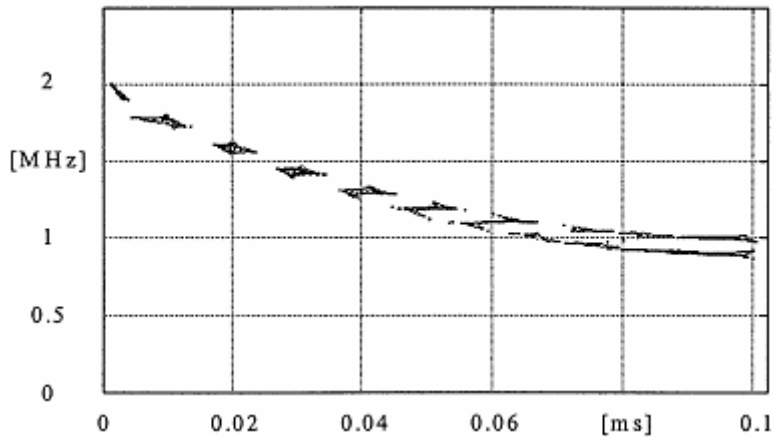

Figure 17: Result obtained from reassigned STFT on a two component test signal (Plot by Angrisani and D'Arco) [7]

V. Adaptive Chirplet Based Signal Approximation

1. Adaptive Approximation

Another method of frequency estimation is proposed by Qian S. and Chen, D. using Adaptive Chirplet combined with the Wigner-Ville distribution [9]. It begins with the assumption that the test signal *s(t)* can be represented as:

$$s(t) = \sum_{p=0}^{p-1} A_p h_p(t) + s_{p+1}(t) \tag{11}$$

where

$s_{p+1}(t) = s_p(t) - A_p h_p(t)$; $s_0 = s(t)$; $A_p = \int s_p(t) h_p(t) dt$. A_p is the inner product of the signal *s(t)* and the chirplet $h_p(t)$; $s_p(t)$ is the residual after the orthogonal projection A_p has been removed from the original signal *s(t)*.

In addition, the chirplet $h_p(t)$ is chosen so that $\min_{h_p} \left\| s_{p+1}(t) \right\|^2 = \min_{h_p} \left\| s_p(t) - A_p h_p(t) \right\|^2$ (12)

This approach guarantees that the residual $\left\|s_{p+1}(t)\right\|^2$ monotonically decreases as the number of terms, *p,* increases. When *p* is large enough and the number of samples is finite, the residual may reduce to zero [9]. In order to reduce the residual, the following equation needs to be solved:

$$\max_{h_p} \left|A_p\right|^2 = \max_{h_p} \left|\int s_p(t) h_p(t) dt\right|^2 \tag{13}$$

However, solving the optimization problem in (13) is difficult. Therefore, $h_p(t)$ has to be limited to be a frequency modulated Gaussian function (14) with $\beta_p = 0$

$$h_p(t) = \left(\frac{a_p}{\pi}\right)^{1/4} e^{-\alpha_p (t-t_p)^2/2 + j(\omega_p t + \beta_p t^2/2)} \tag{14}$$

where α, t_p, ω_p, β are the scaling, time offset, frequency offset, and chirp rate, respectively.

Since the residue $\left\|s_{p+1}(t)\right\|^2$ converges as the number of terms increases, the authors claimed it can be neglected when it is small enough in practical applications [10]. Therefore, (11) can be reduced to:

$$s(t) = \sum_{p=0}^{\infty} A_p h_p(t) \tag{15}$$

The authors then suggested using the zooming algorithm to estimate the optimal chirplet. The zooming algorithm involves the use of a Gaussian function to scan (to project the Gaussian function on the signal) in the frequency and time domain to find the highest projection result. The

deviation and center parameters of the Gaussian function are adjusted until a smallest deviation can be found with the highest peak. Finally, the outputs of the zooming algorithm are:

- The time variance *1/d*, where $d \geq \alpha_p$.
- The center time $t_c = t_p$
- The center frequency $\omega_c = \omega_p$

The Wigner-Ville distribution is then applied to the Gaussian chirplet to obtain a time-frequency representation. Its expression is:

$$WVD_p(t,\omega) = 2e^{-\alpha_p (t-t_p)^2 - (\omega-\omega_p-\beta_p t)^2/\alpha_p} \tag{16}$$

where $WVD_p(t,\omega)$ is the Wigner-Ville distribution of the signal $s(t)$.

Equation (16) is then used to compute the power spectrum of $h_p(t)$. The power spectrum is computed as:

$$\left|H_p(\omega)\right|^2 = \int WVD_p(t,\omega)dt = 2\sqrt{\frac{\pi}{c}}e^{-(\omega-\omega_p-\beta_p t_p)^2/c} \tag{17}$$

In addition, the width of the power spectrum of $h_p(t)$ is proportional to the chirp rate $|\beta_p|$ (the width increases as $|\beta_p|$ increases).

Furthermore, this power spectrum is a Gaussian function which has center frequency at $\omega_c = \omega_p + \beta_p t_p$ (18) (estimated by the zooming algorithm). Its variance is formulated as $\frac{\alpha_p^2 + \beta_p^2}{\alpha_p}$

(19). As a result, when $\beta_p = 0$, the variance is $\alpha_p = d$ where $1/d$ is the time variance estimated by the zooming algorithm [9].

2. Computer Algorithm

The authors suggested that the adaptive approximation algorithm can be implemented as follow:

a. Apply the zooming algorithm to estimate the time variance t/d, center time t_c, and center frequency ω_c.

b. Compute the signal's power spectrum using (17).

c. Estimate the frequency variance of the Gaussian function center at ω_c.

d. If the variance equals to d, then $\beta_p = 0$, $\alpha_p = d$ =variance, $t_c = t_p$, $\omega_c = \omega_p$. If not, go to step e.

e. Set the initial value of α_p to d.

f. Compute β_p (chirp rate) by (19) and ω_c (center frequency) by (18).

g. Construct $h_p(t)$ with found parameters using (14).

h. Compute the inner product of $h_p(t)$ and $s_p(t)$ using (15).

i. Reduce α_p by a small quantity of $\Delta\alpha$ ($\alpha_p = \alpha_p - \Delta\alpha$) and repeat step f to step h until $\alpha_p = 0$. The parameters (α_p, t_p, ω_p, β_p) corresponding to the largest $|A_p|^2$ constitute the optimal chirplet $h_p(t)$.

The negative outcome of this algorithm above is that it can be very computationally intensive which may not be suitable for embedded processors with real-time processing. As described above, step *i* is the most time consuming because of the iterative decrement of parameter α_p and the estimation accuracy depends on the amount of $\Delta\alpha$. However, because $|A_p|^2$ is less sensitive to small α_p, $\Delta\alpha$ can be gradually increase as α_p gets smaller [9].

3. Testing and Results

Tests on this method were done by the authors on an echo-location pulse test signal emitted by a large brown bat. The signal was digitized into 400 samples and its plot in the time domain is in Figure 18. Subsequently, the relative residual used for approximation is defined as $residual = \frac{\|s_p(t)\|^2}{\|s(t)\|^2} \times 100$.

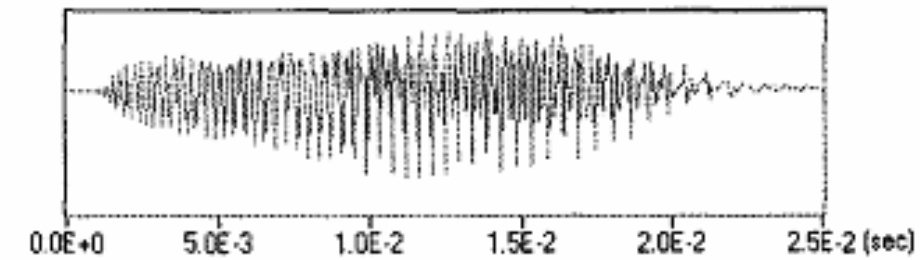

Figure 18: Test signal *s(t)* of a bat sound used for approximation (Image by Qian and Chen) [9]

As shown the image, the test signal is clearly a chirp signal with a nearly smooth Gaussian envelop which is perfect for the purpose of testing this Adaptive Approximation method. The expected result is also expected to be smooth with negative slope in the time-frequency plot. The authors subsequently performed the tests on a Pentium 120 PC [9]. The test signal *s(t)* was approximated and represented by five chirplets. The test then reportedly ran for 0.49 second producing final residuals of less than ten percent. Those residuals obtained from each chirplet are listed in Table 1. On the other hand, the time-frequency plot of the result obtained from this method is illustrated in Figure 19. The representation of the chirps in the results seems to be very smooth and well-separated as two distinct components. Nevertheless, to obtain this result, the Chirplet-based Adaptive approach took a signification amount of time [9] because of the computationally intensive operations on the zooming algorithm initially (Step a) and parameter searching in Step i.

Number of Chirplets	Residual (%)	Processing time (second)
1	58.0	0.11
2	31.1	0.22
3	22.7	0.29
4	15.9	0.38
5	9.76	0.49

Table 1: Residuals obtained from adaptive approximation [9]

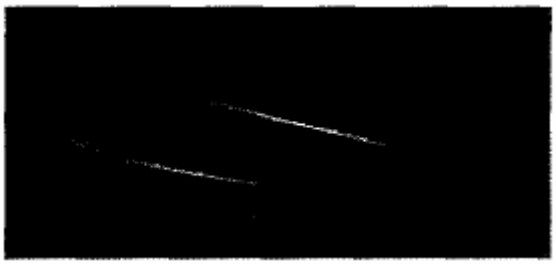

Figure 19: Spectrogram of results obtained with Chirplet Based Adaptive Approximation (5 chirplets) (Image by Qian and Chen) [9]

CHAPTER 2

PROPOSED METHOD

a. Projection Formulation

According to the analysis and performance of the STFT method, the modified Chirplet Transform, and the Adaptive Approximation method, the approach of projecting basis functions, derived from a mother chirplet, on the analyzed signal seems to be the most accurate. For all methods, the choice of the window size is important because it determines the time and frequency resolution on the time-frequency plane, and thus the accuracy in the instantaneous frequency representation. The window size must also relate to the period of the signal under consideration in order to be able to capture accurately information from all frequency components. More specifically, it is important to have a window size of as least equal to the period of the signal's lowest frequency component (slowest component – largest period).

The process of finding the maximum inner-product between the signal and the projected basis functions, when these functions are chosen arbitrarily, is similar to "guessing" the spectral characteristics of the signal. Multiple projections can be performed on a signal segment at a particular time, and the projection that results in the highest inner-product is the one that gives the best estimate of the signal properties for that particular segment. However, in practical applications, and specifically, using embedded processors, there is a trade-off between accuracy

and real-time processing. Thus, the proposed algorithm must be adaptable to the different hardware speed and resources.

Instead of projecting a family of chirplets onto the original signal, our proposed method used short basis functions, namely cosine and sine segments, to approximate the spectral components locally by maximizing the projection outputs. The proposed method assumes that in a significantly short window of time, the spectral components of the signal do not change rapidly. As described in the equation next, these cosine basis functions are combined with same-frequency and same-length sine functions. Both cosine and sine segments are multiplied by a Gaussian window. The combination of cosine and sine has the effect of averaging the projection by taking the magnitude of the signal's real and imaginary components. Projecting is done in an iteratively manner starting from the beginning up to the end of the signal segment, in order to identify the location of matching frequencies, if any. Since in real-time systems, signals are usually buffered in small segments, then processed, and finally transmitted to another processing block, using this iterative projection approach is reasonable. The estimated frequency f_o is given by:

$$f_o = \arg\max_{i,n}\left\{\int_{-\infty}^{\infty}\left[s(\tau-t)\cdot w(t-n)\cdot C_i(t-n))\right]^2+\left[s(\tau-t)\cdot w(t-n)\cdot S_i(t-n))\right]^2 dt\right\} \qquad (20)$$

where $C_i(t) = \cos(2\pi f_i t)$, $t = 0,\ldots, kT_i$, $S_i(t) = \sin(2\pi f_i t)$,), $t = 0,\ldots, kT_i$, k is an integer, T_i is equal to the period of $\cos(2\pi f_i t)$ and $\sin(2\pi f_i t)$, i denotes the i^{th} frequency examined, $w(t)$ is a Gaussian envelope, and n represents the shift. Essentially, the lengths of $C_i(t)$ and $S_i(t)$ are equal to multiples of the period of $\cos(2\pi f_i t)$ and $\sin(2\pi f_i t)$.

In summary, the above equation states that a frequency of a signal at any point in time can be estimated by projecting a several different-frequency cosine/sine basis functions on windowed signal segments. The frequency that maximizes the combined projections is the best-estimated frequency for that particular time instance. Although both cosine and sine basis functions are needed, using cosine basis function only is often adequate for frequency estimation. The combination of cosine and sine basis functions results in a smoother trajectory.

There are three parameters that can be modified to adapt to hardware speed and resources: window size, frequency range, and the amount of time shift. Unlike the STFT, whose most appropriate window size cannot be easily predicted, the proposed technique allows window sizes to change in a very specific manner. Trade-offs between accuracy and performance can also be done by adjusting the time shift parameter τ and the resolution of frequency range. Specifically, adjusting the time shift amount would result in different number of projections.

b. Computer Algorithm

To efficiently compute frequency estimation using our method in embedded processors, the following steps are proposed:

Generate a table of a predefined range of sine and cosine basis functions. Decide on an estimation range of frequencies. Then for each frequency in the range, generate a cosine and/or sine base function of that frequency.

Load the table of basis functions into memory during initialization. The elements of each coefficient in the table are then accessed by indexing at the time of processing.

Iteratively projecting the basis functions on the segments of the signal being analyzed as described in (20).

Using peak detection algorithm to find the highest peak from the projection results with the indexes of the highest projections are the estimated frequencies. Create a map of time and frequencies estimated for later reconstruction.

CHAPTER 3

RESULTS AND ANALYSIS

The first part of testing is done in MATLAB, by simulating the algorithm performance in Personal Computer using Intel Pentium 4 processor running at 2.0 Gigahertz (GHz) with 1 Gigabyte (GB) of memory. Testing is done on both single and multiple spectral component signals. Subsequently, the effect of each of the key parameters, window size, time shift, and frequency range, on the algorithm's performance is also demonstrated. The results of these tests are plotted on time-frequency planes for illustration purposes. Finally, these results are compared to results obtained from STFT applied on the same test signals. Analysis on the results for both techniques is also carried out to conclude whether the proposed method is more efficient and/or accurate than STFT. The MATLAB code for testing is included in Appendix A.

The following MATLAB function is used for testing with STFT (MATLAB Help Documentation):

```
spectrogram(x,window,noverlap,nfft,fs)
```

window: a Hamming window of length nfft.

noverlap: the number of overlapping segments that produces 50% overlap between segments.

nfft: the FFT length and is the maximum of 256 or the next power of 2 greater than the length of each segment of x.

fs: sampling frequency in Hz.

I. Test on single linear spectral component signal

The resulting projection curve shown in Figure 23 is used to find at which time instance each basis function provides the highest peak. As a reminder, each basis function is a cosine or sine segments. Therefore, if a basis function provides a high peak at a particular instance, it is implied that the instantaneous frequency at that point in time is similar to that at the projected cosine/sine. Peak detection is applied to search for highest matching frequencies. If more than one peak with high amplitude is found at the same time instance by projecting different basis functions, the signal is determined to have multiple components. Then a multi-peak detection combined with tracking algorithm is necessary to find each component in the signal. The test signal in Figure 20 has only one component which results in only one peak being detected. However, the plot of the reconstructed spectral trajectory appeared to be non-continuous because of the use of a scanning interval of 10Hz. The use of frequency scanning intervals reduces a tremendous amount of computations needed for detecting chirps but also degrades the frequency resolution on the time-frequency plane. As a result, when computing resources are scarce, increasing the frequency scanning intervals provides a clear advantage. However, the designer must be cautioned that only when chirps are linear and slowly changed, large intervals are appropriate. Otherwise, when chirps are fast changing, either line regression or smaller scanning interval must be used to maintain accuracy of the results.

Figure 25 illustrates the result from the same signal but with smaller frequency scanning intervals and larger time interval τ. The detected points are obviously denser when using small search frequency intervals. Both figures, however, display very consistent results, a straight line with slope of 0.001 starting at 200 Hz. Result from Figure 24 is obtained with a smaller window size (parameter k) of 4 times the period of the searching frequency i versus 10 times as in Figure 24. Finally, the results are compared against STFT with an FFT size of 1024 which produces a similar curve to the proposed method's result. The result from the STFT is obtained by using MATLAB which produces an image-type plot. Because of its high resolution in time scale, the time-frequency curve appears to be thick. However, peak detection can also be applied here to obtain a trajectory.

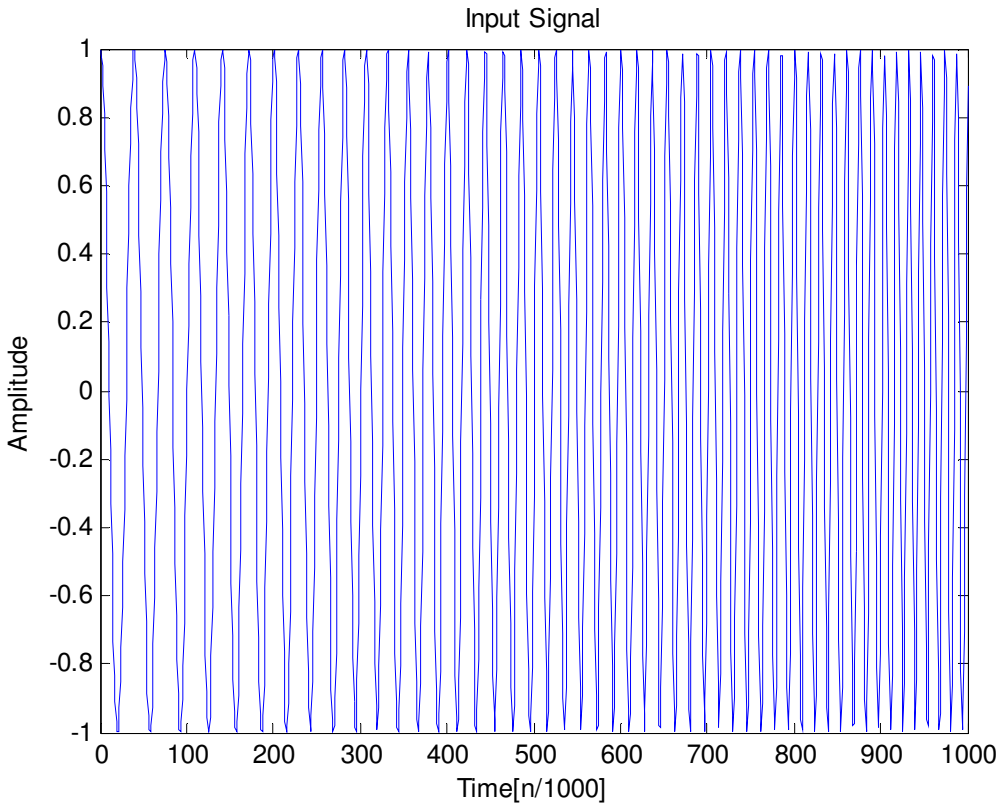

Figure 20: Test signal with linear chirp rate = 0.001 start at 200 Hz, Sampling Frequency = 8000

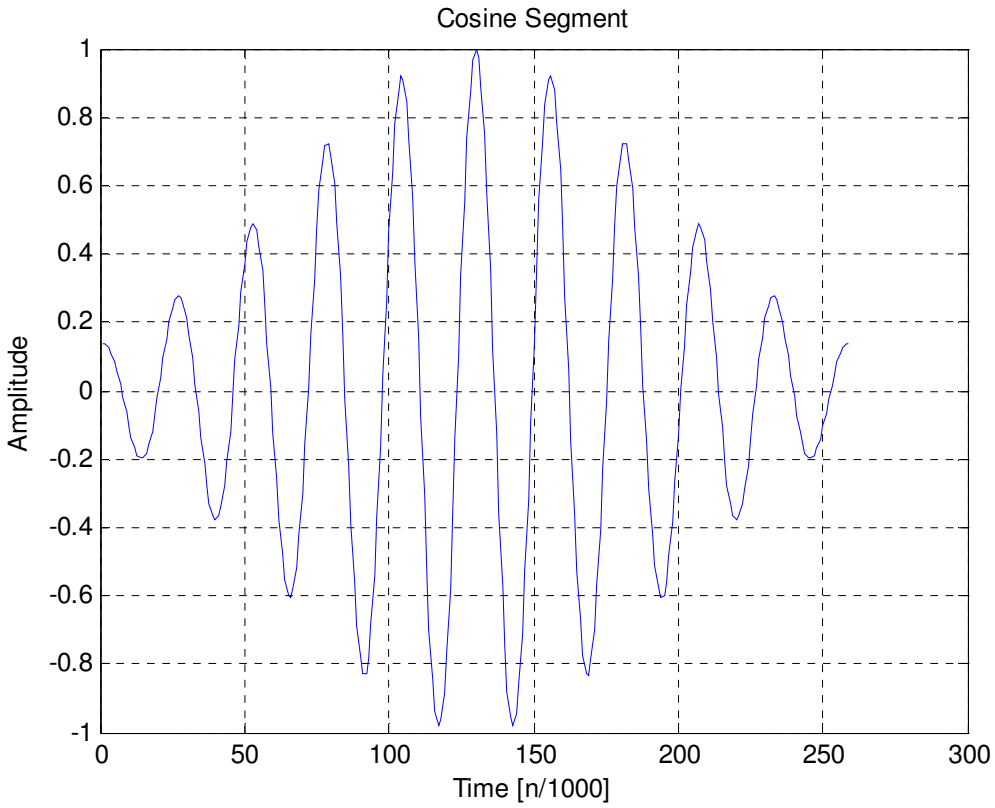

Figure 21: Example of Gaussian Windowed Cosine Segment Used for Projection

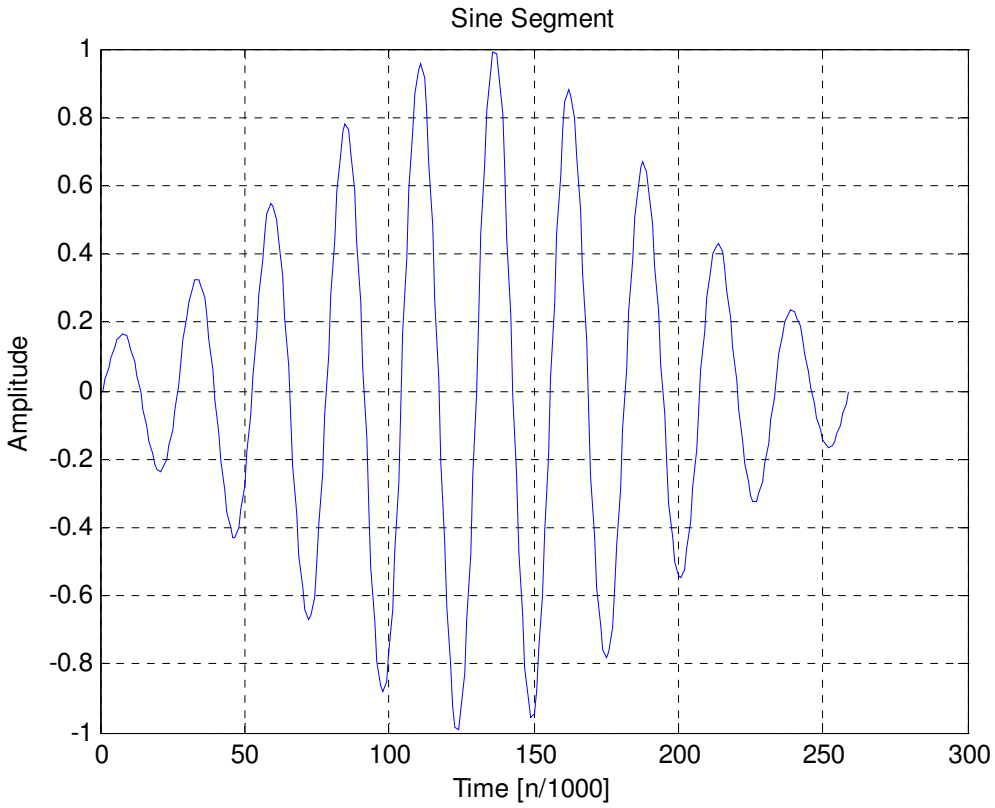

Figure 22: Example of Gaussian Windowed Sine Segment Used for Projection

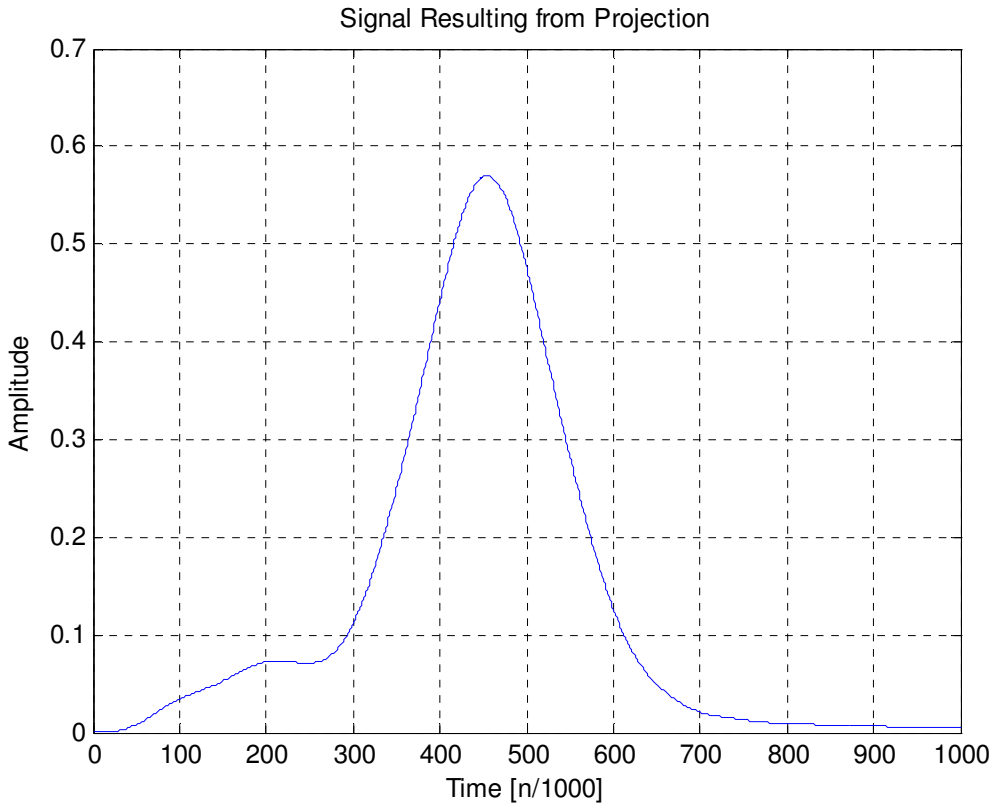

Figure 23: Example of Signal Resulting from Projection Used for Peak Detection. (The highest peak indicates matching frequency found. Other low peaks will be discarded by applying a threshold)

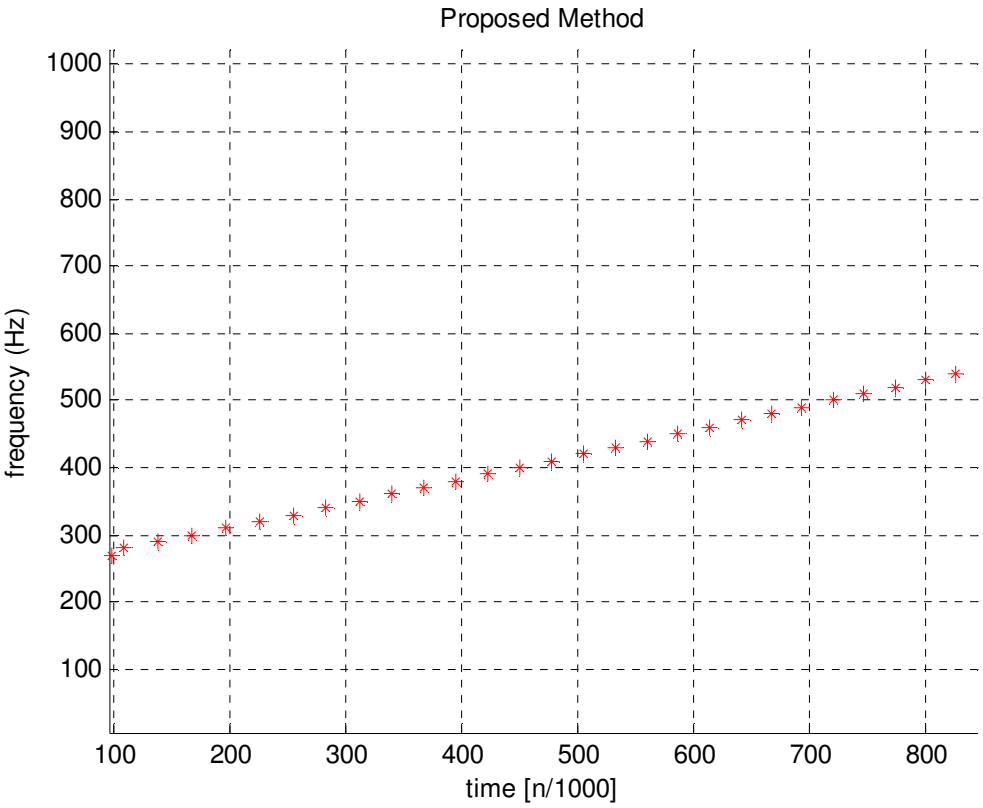

Figure 24: Results from Proposed Method with Peak Detection (Window Size = 10*Period of i, $\tau = 1$ Scanning Frequency Range i = 100-4000Hz with Interval of 10Hz)

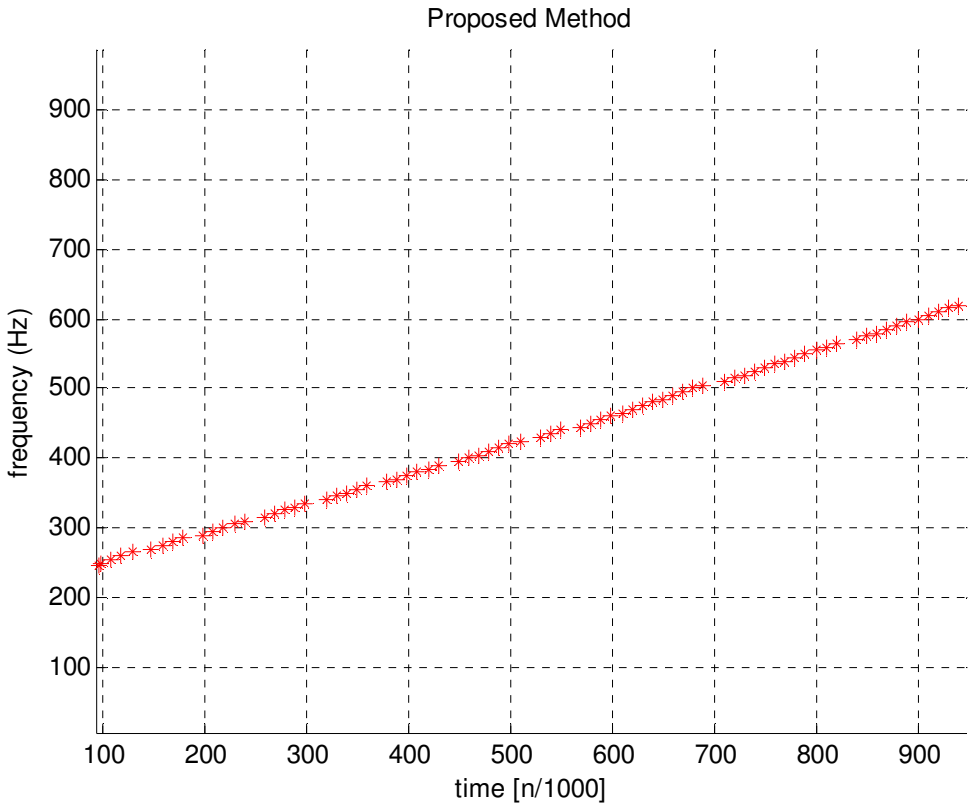

Figure 25: Results from Proposed Method with Peak Detection (Window Size = 4*Period of i, τ = 10 Scanning Frequency Range i = 100-4000Hz with Interval of 5Hz)

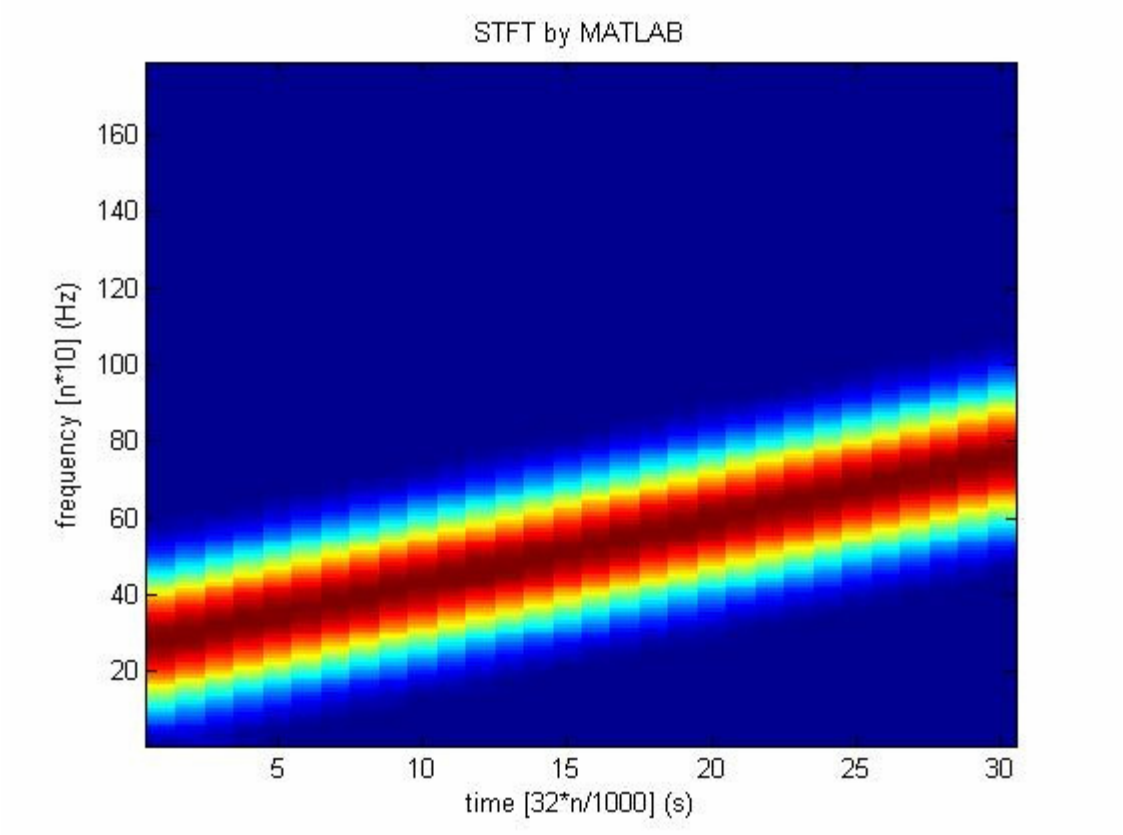

Figure 26: Results from STFT (Spectrogram) by MATLAB (Window Size = 64, FFT size = 1024, 50% overlapped)

II. Test on single non-linear spectral component signal

In this test, a quadratic chirp is generated starting at 300Hz. Applying the proposed method, a smooth and accurate trajectory is detected and reconstructed in both cases of using large frequency interval (100Hz) and small frequency interval (5Hz). The result from using small frequency interval, however, is smoother due to the higher frequency resolution. On the other hand, STFT's trajectory is not smooth in the case of using large window size (64 samples) and 50% overlapping. The trajectory is smoother when using small window size (32 samples) and 100% overlapping.

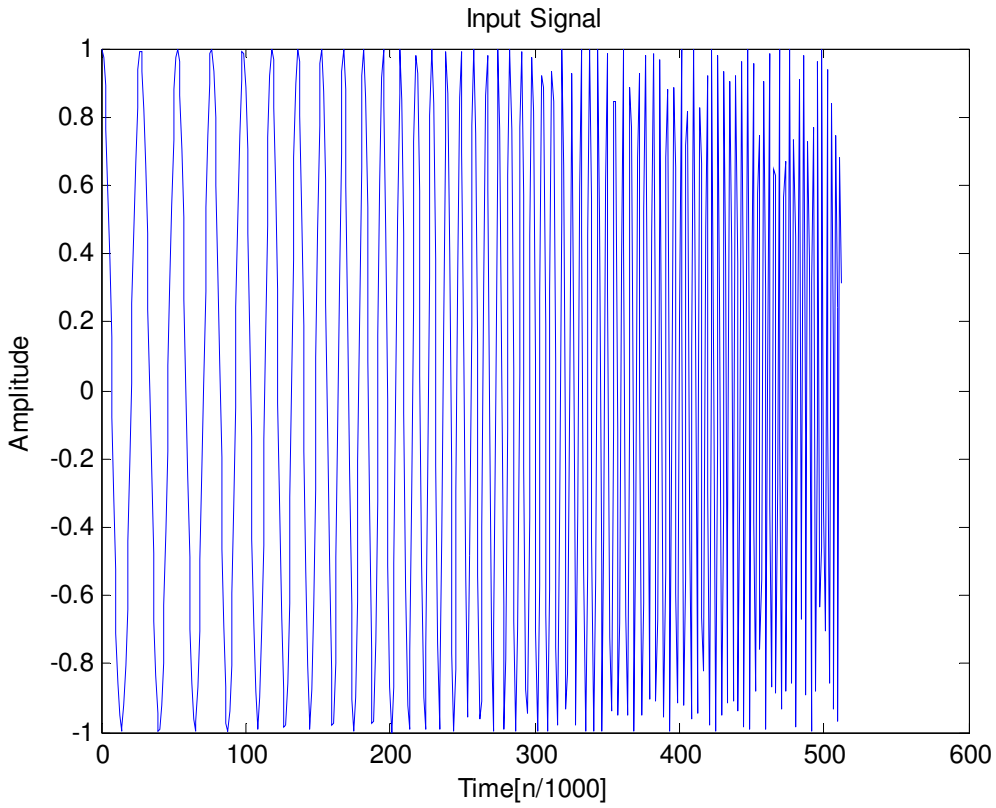

Figure 27: Test signal with quadratic chirp rate = 0.00001 start at 300 Hz, Sampling Frequency = 8000

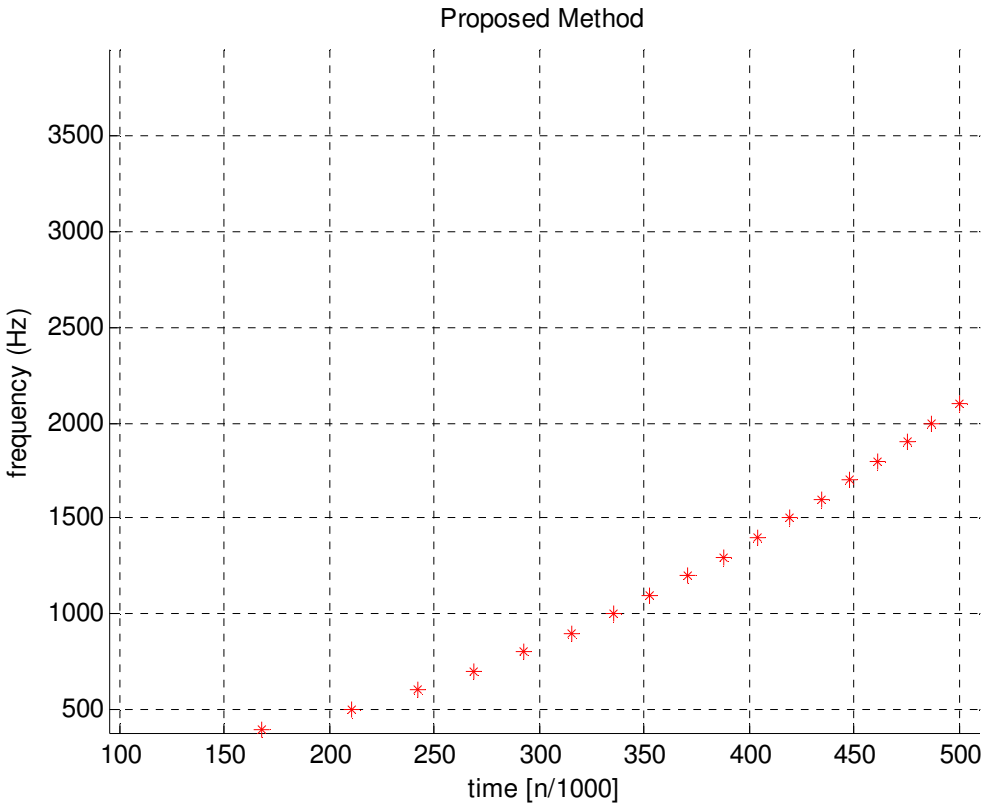

Figure 28: Results from Proposed Method with Peak Detection (Window Size = 2*Period of i, $\tau = 1$, Scanning Frequency Range i = 100-4000Hz with Interval of 100Hz)

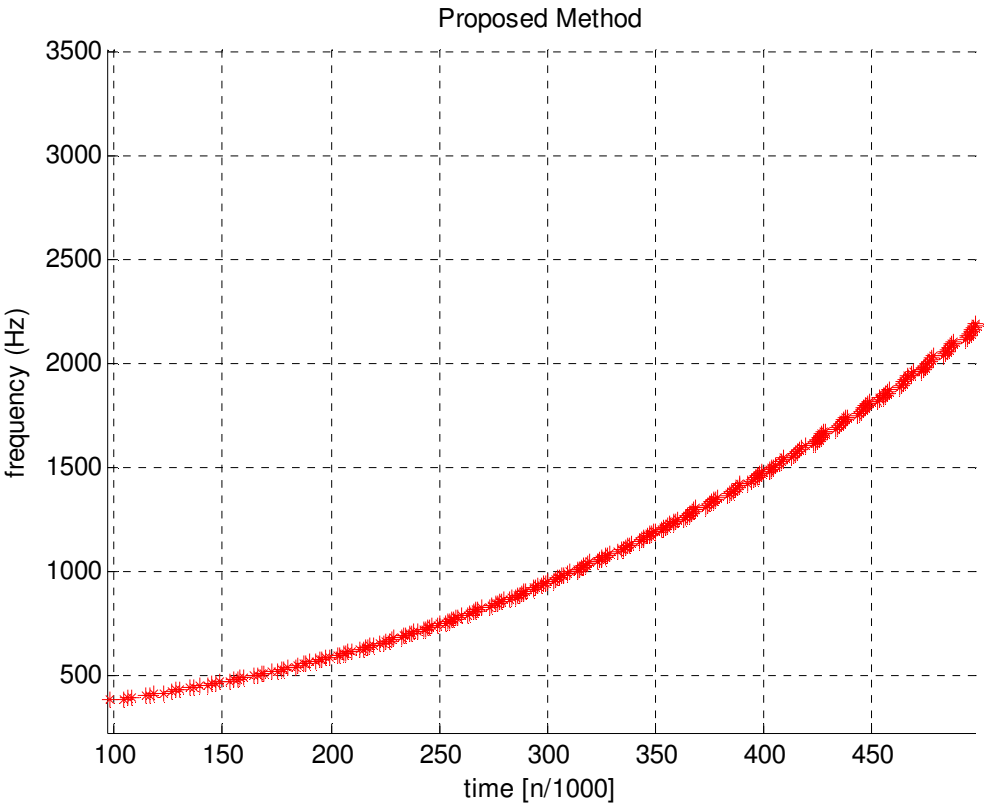

Figure 29: Results from Proposed Method with Peak Detection (Window Size = 4*Period of i, τ = 10, Scanning Frequency Range i = 100-4000Hz with Interval of 5 Hz

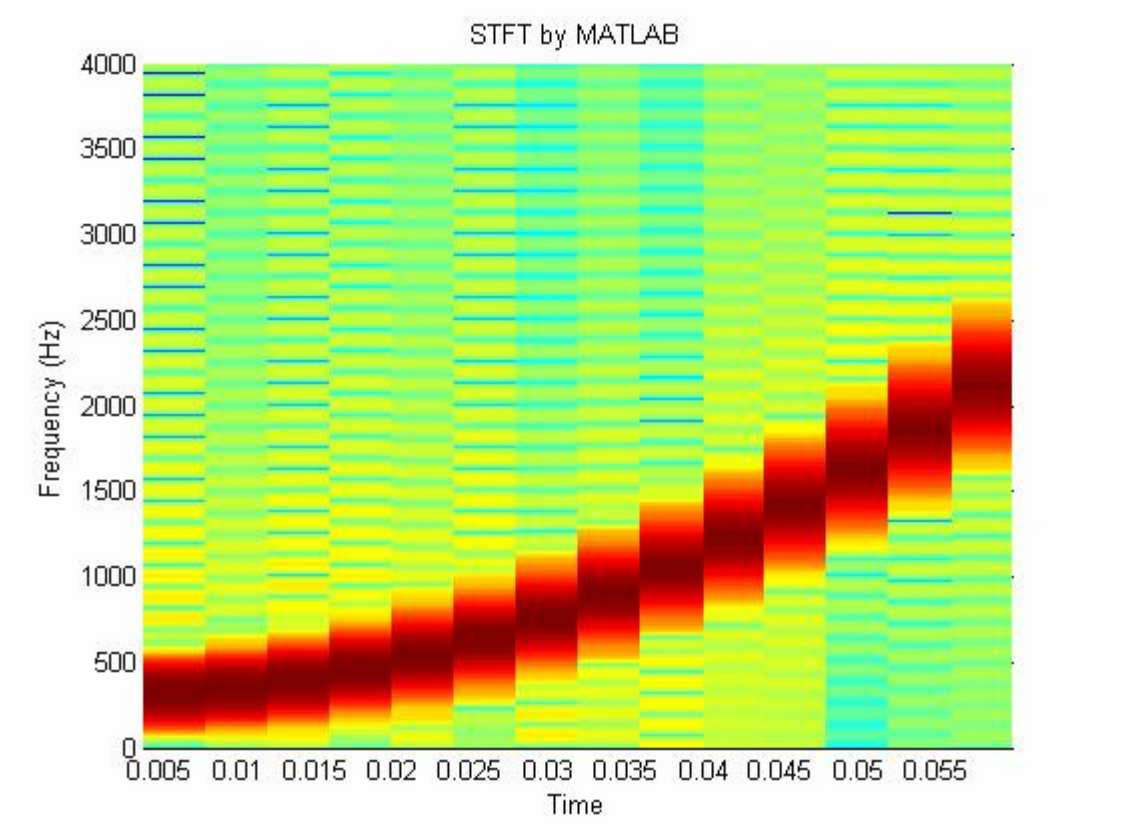

Figure 30: Results from STFT (Spectrogram) by MATLAB (Window Size = 64, FFT size = 1024, Overlapped Size = 32)

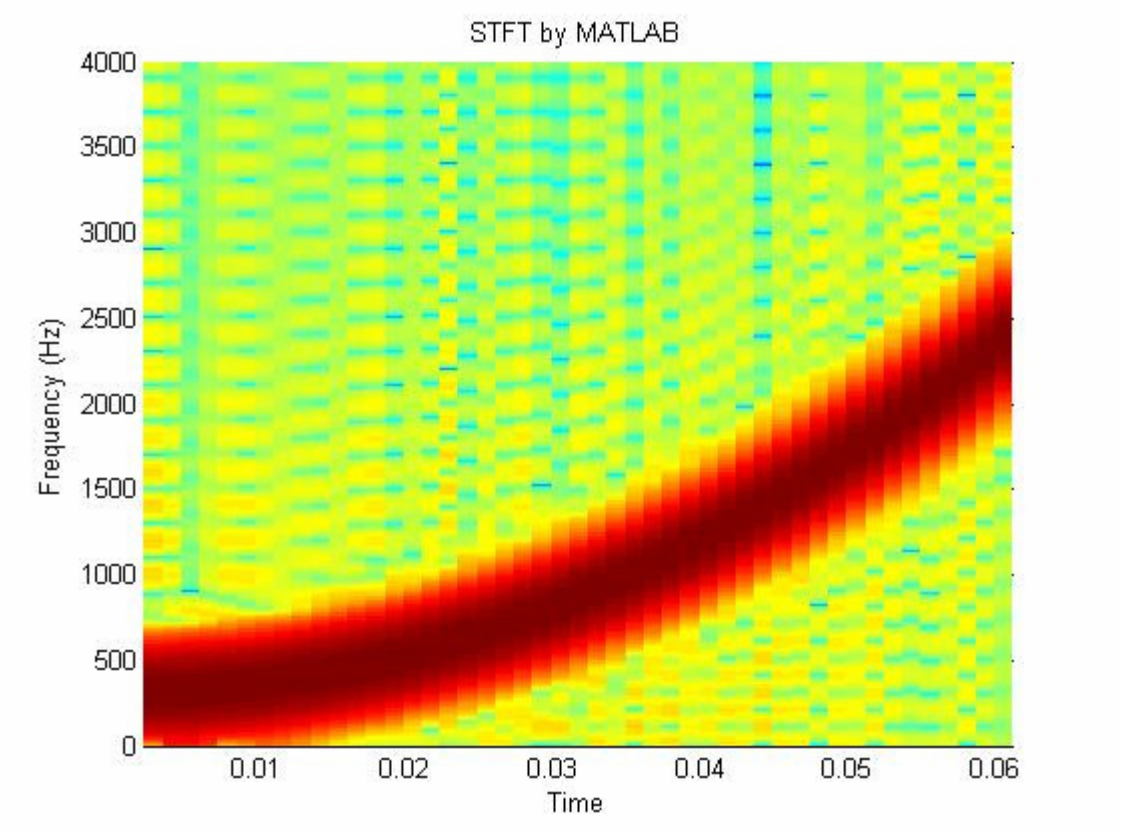

Figure 31: Results from STFT (Spectrogram) by MATLAB (Window Size = 32, FFT size = 1024, Overlapped Size = 30)

III. Test on single non-linear fast-changing spectral component signal

In this test case, the signal is generated with quadratic chirp rate that is twice as fast as the previous case (0.00002). The time shift parameter τ of the proposed method is varied to demonstrate its effects on the results. Figure 37 is the result from STFT which shows an almost linear trajectory. Figure 34 and Figure 35 are results from proposed method with a time shift of 10 samples in both cases. However, in Figure 33, only the cosine part of (20) is used which leads to non-continuous trajectory. In Figure 34, trajectory resulted from using widow sizes that are 4 times the projecting periods is smoother that of using window size that are 2 times the projecting periods (Figure 36).

Figure 33 also shows that trajectory resulted from using 100Hz interval scanning frequency is more un-continuous. Consequently, when computing resources are limited, to maintain reasonable accuracy, the most important consideration should be to increase the time-shift parameter τ to reduce computational time.

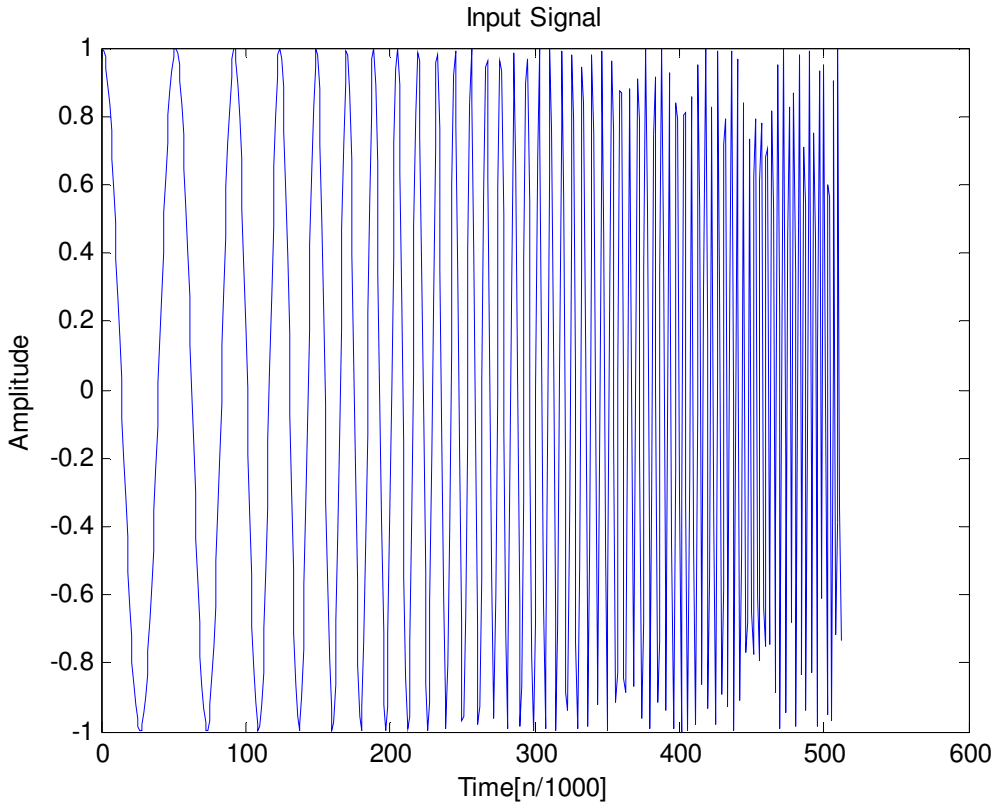

Figure 32: Test signal with quadratic chirp rate = 0.00002 start at 300 Hz, Sampling Frequency = 16000 Hz

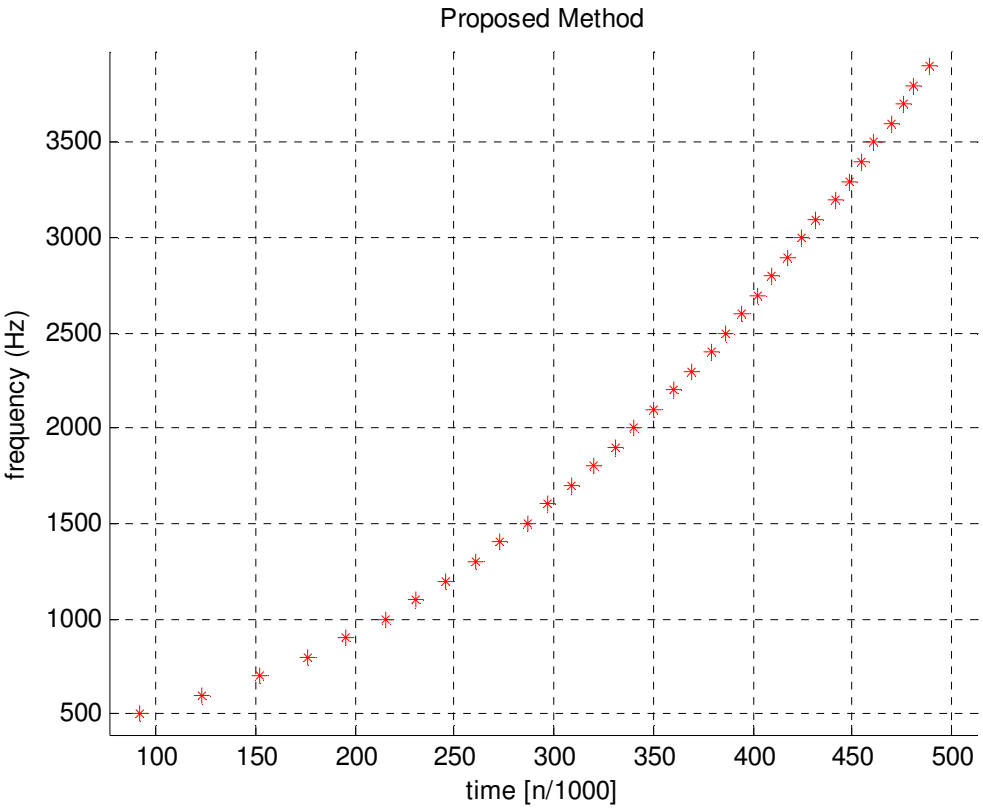

Figure 33: Results from Proposed Method with Peak Detection (Window Size = 4*Period of i, $\tau = 1$, Scanning Frequency Range i = 100-4000Hz with Interval of 100 Hz, Cosine Projection Only)

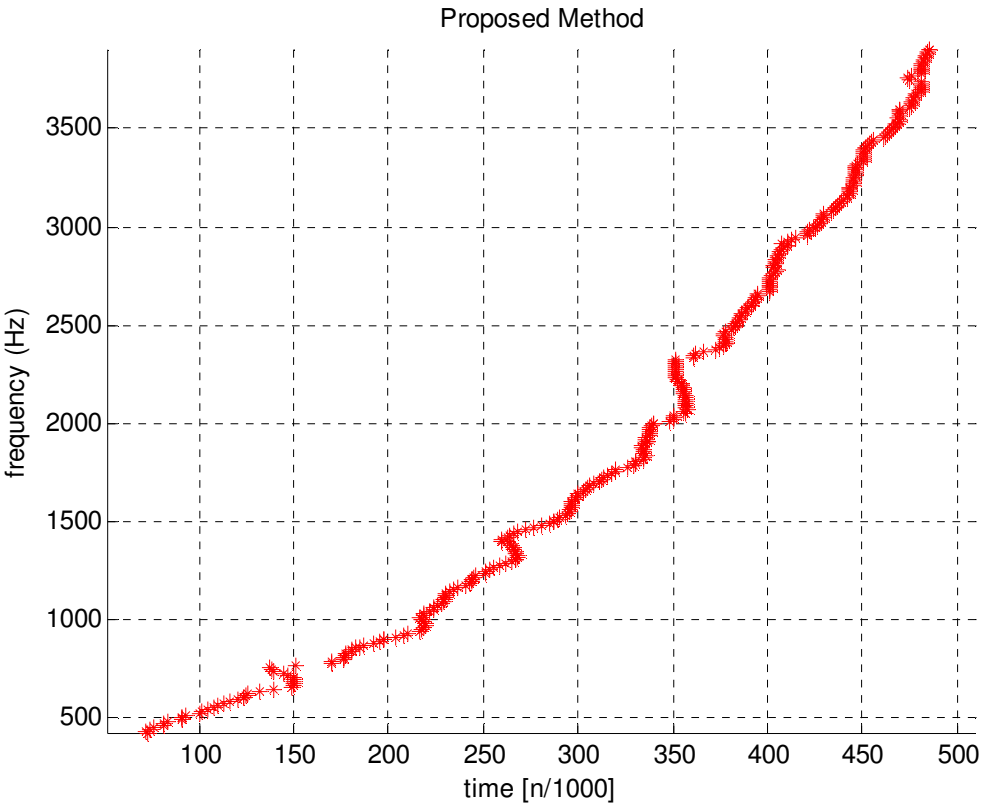

Figure 34: Results from Proposed Method with Peak Detection (Window Size = 4*Period of i, τ = 10, Scanning Frequency Range i = 100-4000Hz with Interval of 10 Hz, Cosine Projection Only)

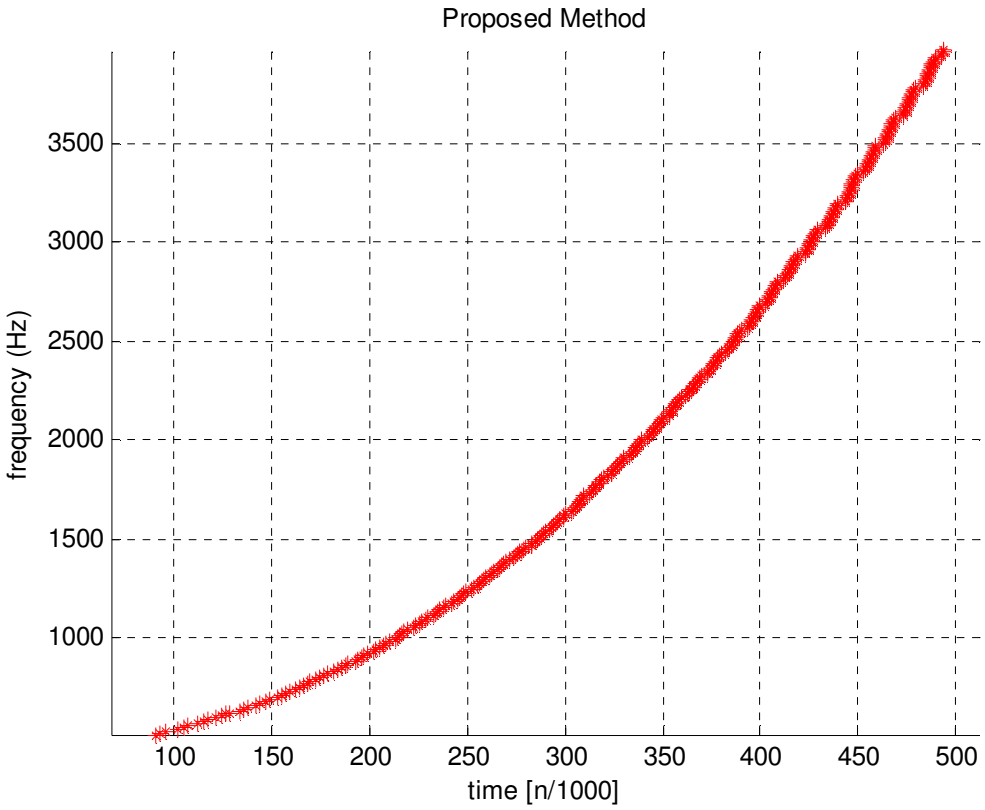

Figure 35: Results from Proposed Method with Peak Detection (Window Size = 4*Period of i, τ = 10, Scanning Frequency Range i = 100-4000Hz with Interval of 10 Hz, Cosine and Sine Projection)

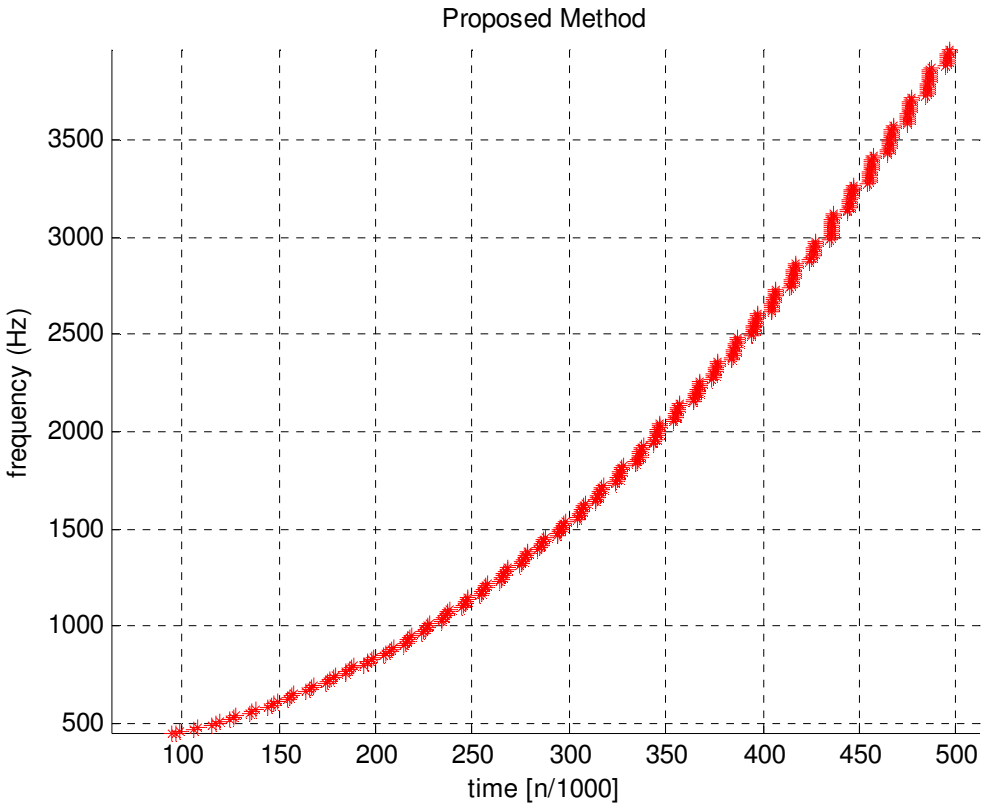

Figure 36: Results from Proposed Method with Peak Detection (Window Size = 2*Period of i, τ = 10, Scanning Frequency Range i = 100-4000Hz with Interval of 10 Hz, Cosine and Sine Projection)

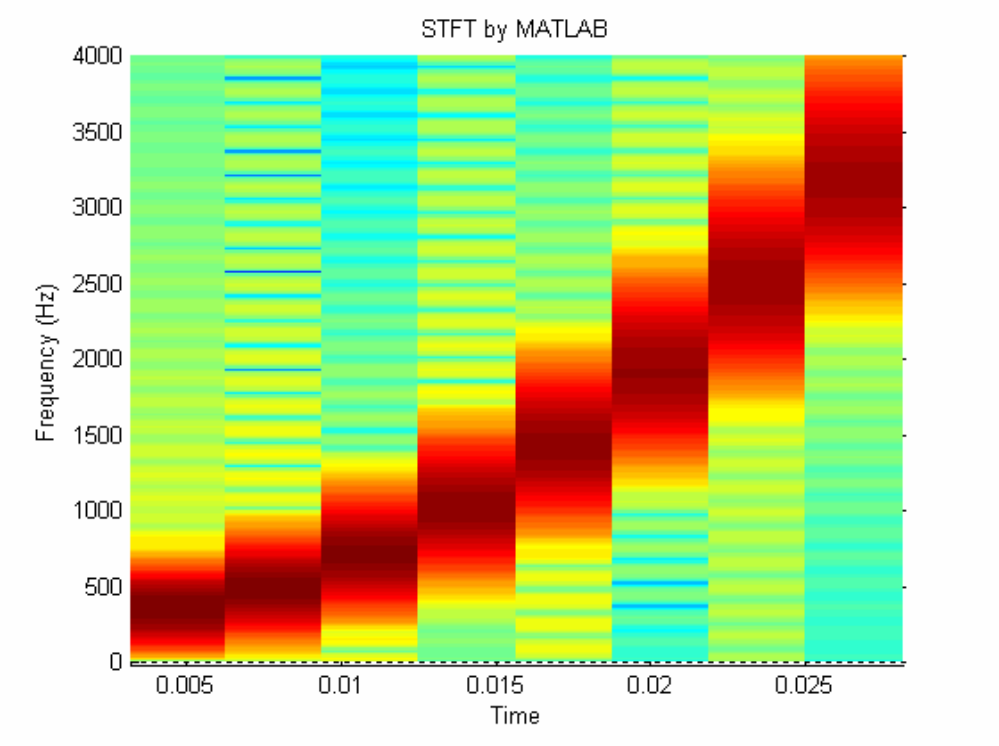

Figure 37: Results from STFT (Spectrogram) by MATLAB (Window Size = 100, FFT size = 1024, Overlapped Size = 50)

IV. Test on multiple linear spectral component signal

With multiple components crossing each other, the results from both STFT and the proposed method produce similar trajectories. No spectral components are detected at time 0.03s using the proposed method because the two components of the signal combine destructively.

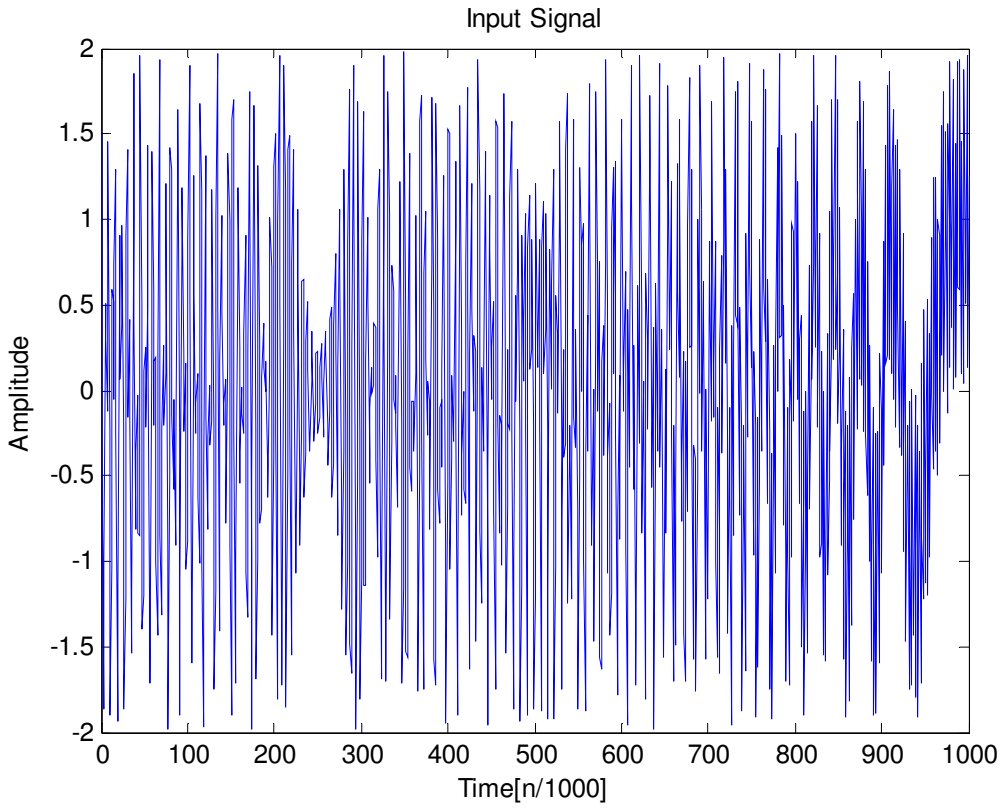

Figure 38: Test signal with linear chirp rate = 0.001 start at 1000 Hz, Sampling Frequency = 8000 Hz

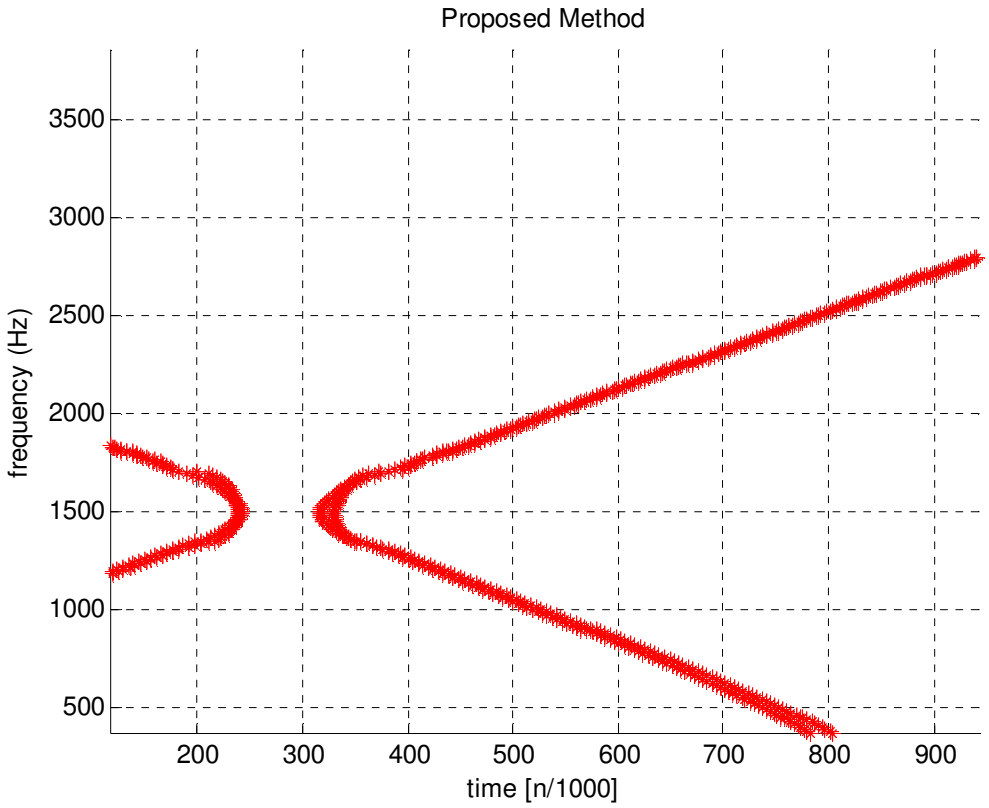

Figure 39: Results from Proposed Method With Peak Detection (Window Size = 6*Period of i, $\tau = 1$, Scanning Frequency Range i = 100-4000Hz with Interval of 10 Hz, Cosine and Sine Projection)

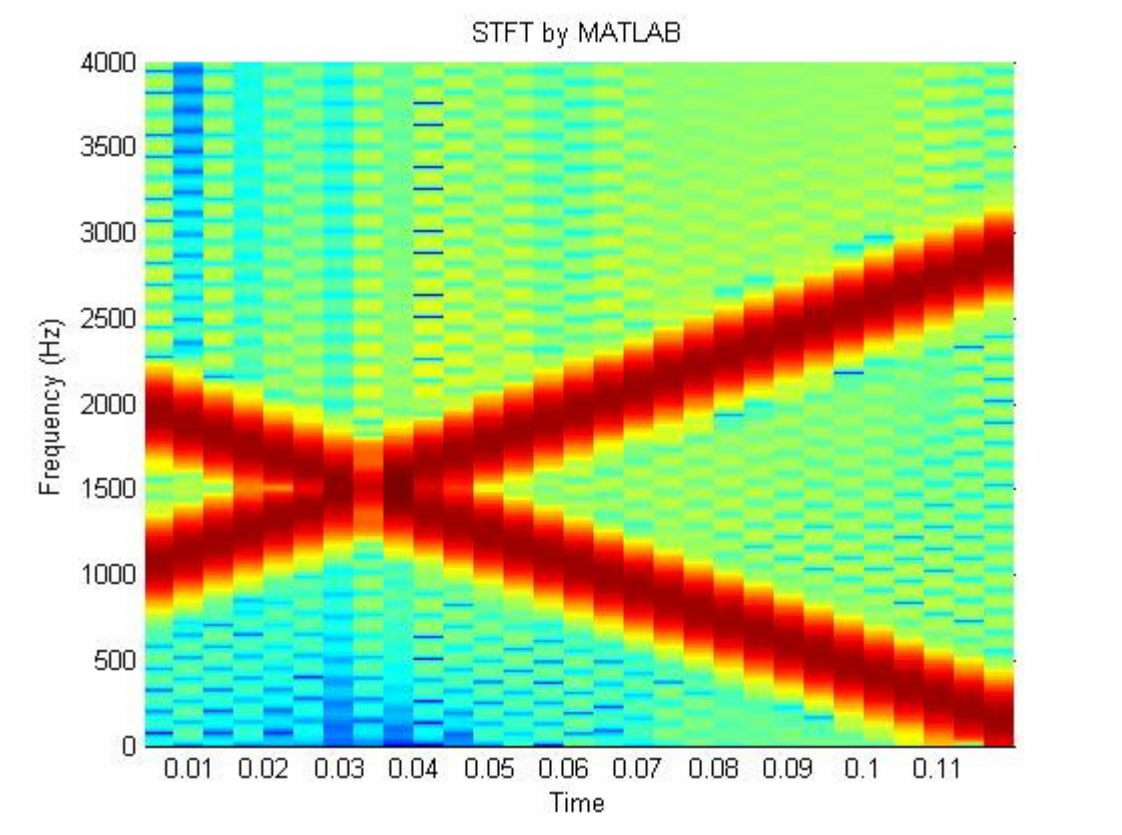

Figure 40: Results from STFT (Spectrogram) by MATLAB (Window Size = 64, FFT size = 1024, Overlapped Size = 32)

V. Test on multiple linear non-crossing spectral component signal

For linear chirping with both components chirping non-crossing (non-destructive) to each other, both STFT and the proposed method produce similar trajectories. The proposed method is able to track two distinguishable paths for both components.

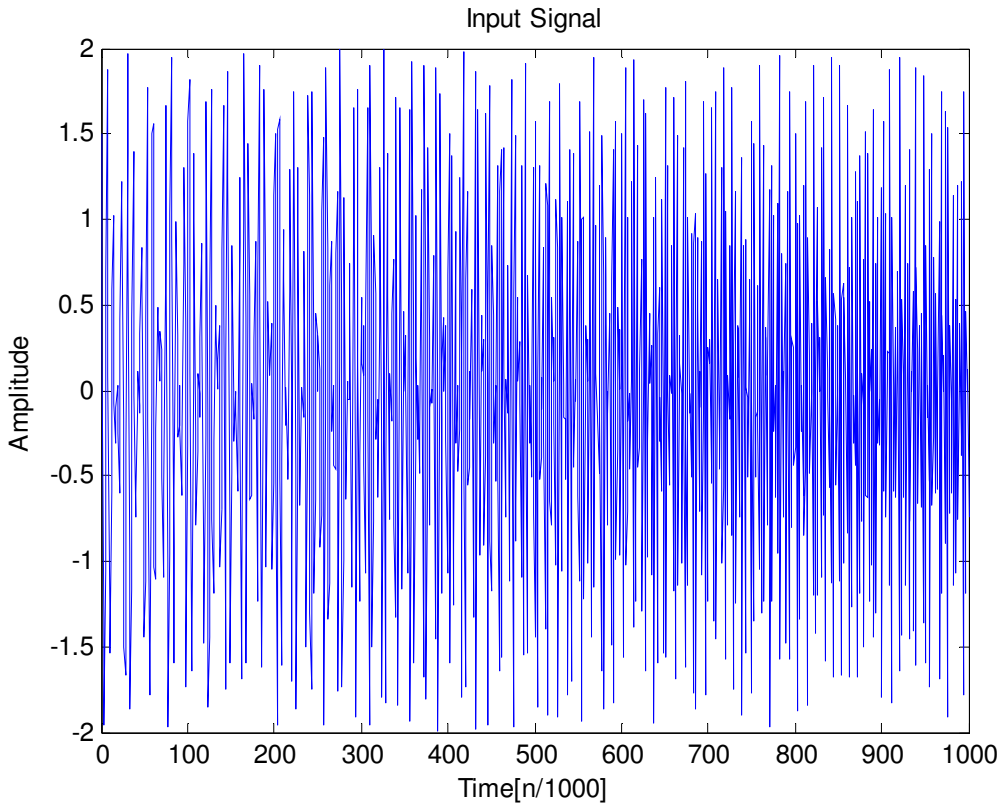

Figure 41: Test signal with linear chirp = 0.001 start at 1000 Hz and 1300 Hz, Sampling Frequency = 8000 Hz

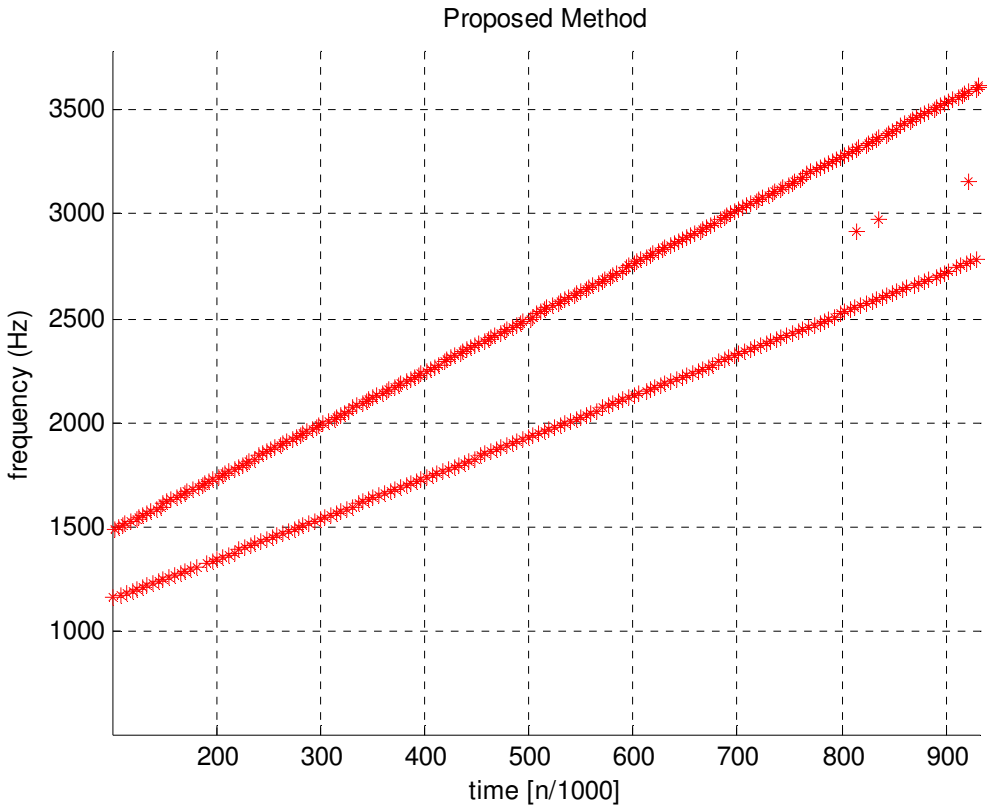

Figure 42: Results from Proposed Method with Peak Detection(Window Size = 8*Period of i, τ = 1, Scanning Frequency Range i = 100-4000Hz with Interval of 10 Hz, Cosine and Sine Projection)

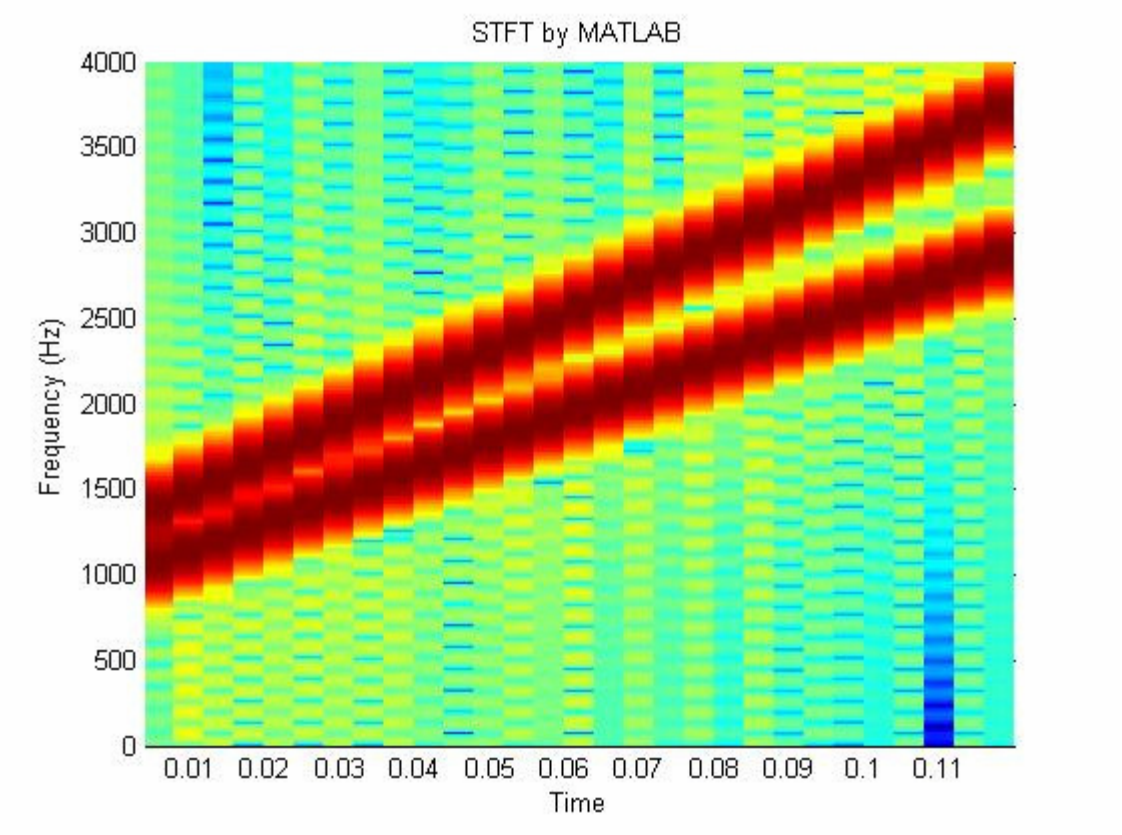

Figure 43: Results from STFT (Spectrogram) by MATLAB (Window Size = 64, FFT size = 1024, Overlapped Size = 32)

VI. Test on multiple linear and quadratic spectral component signal

In this test case, a linear and a quadratic chirp are combined. Aliasing is purposefully allowed to occur at approximately 0.45s which produces a fast downward chirp component (Figure 44). The proposed method performs acceptable for linear and quadratic chirp in this case. Figure 44 also shows the input signal in the time domain with almost zero-amplitude approximately at time 0.015s because the two signal components add destructively. This is the reason why there is almost no frequency component detected at that time for both the STFT and the proposed method. In the trajectory plot (Figure 45), the result for the proposed method shows only one component prior to

this time because of the fact that the two components are too close together which make peak detection very difficult. Both window sizes used for the STFT are no able to produce acceptable results. This emphasizes the fact that an appropriate STFT window may not be easy to find. Figure 46 displays the results of STFT when using a window size that is too small (16 samples) which is barely acceptable for the quadratic component but is unacceptable for the linear component. On the other hand, Figure 47 is the result of using window size that is too large (128) which produces acceptable result for the linear component but unacceptable result for the quadratic component. In practice, therefore, it is difficult to estimate the optimum window size in the case of using STFT unless an optimal size is iteratively selected after FT is done which needs more intensive calculations. In contrast, the proposed method uses window sizes which are immensely proportional to the searching frequencies. This implies that the window sizes adapt to the frequencies of the particular basis used. Therefore, the trajectories for both components are visible (Figure 48).

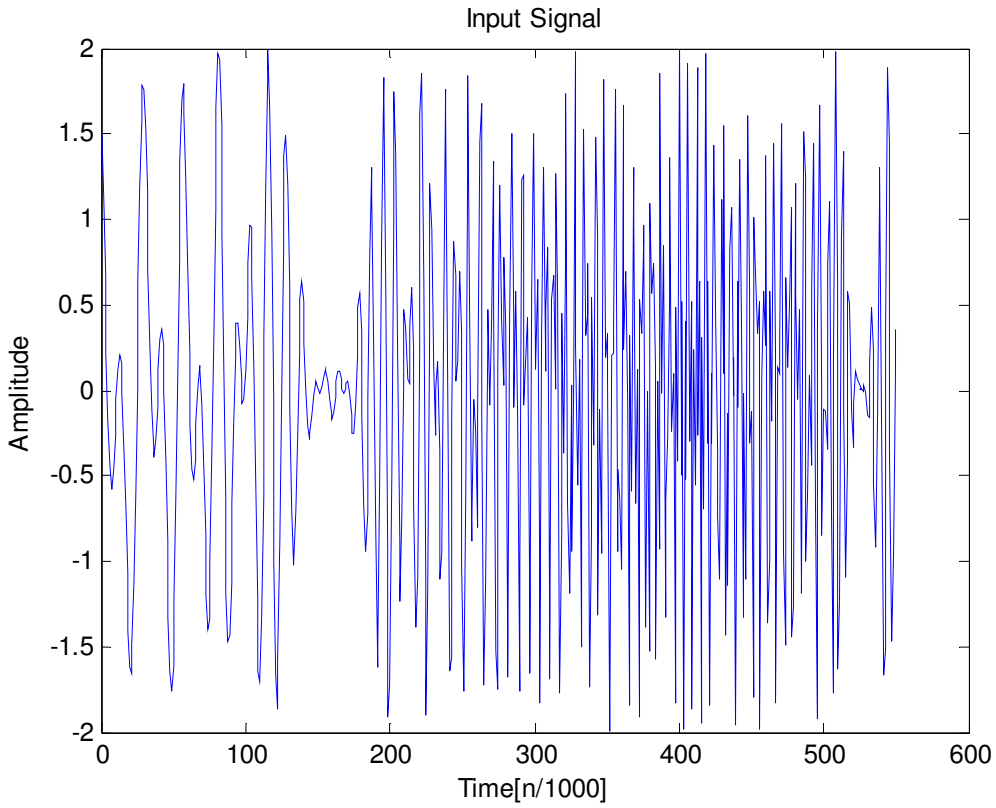

Figure 44: Test signal with linear chirp = 0.002 and quadratic chirp = 0.00003 start at 1000 Hz and 500 Hz, respectively. Sampling Frequency = 16000 Hz

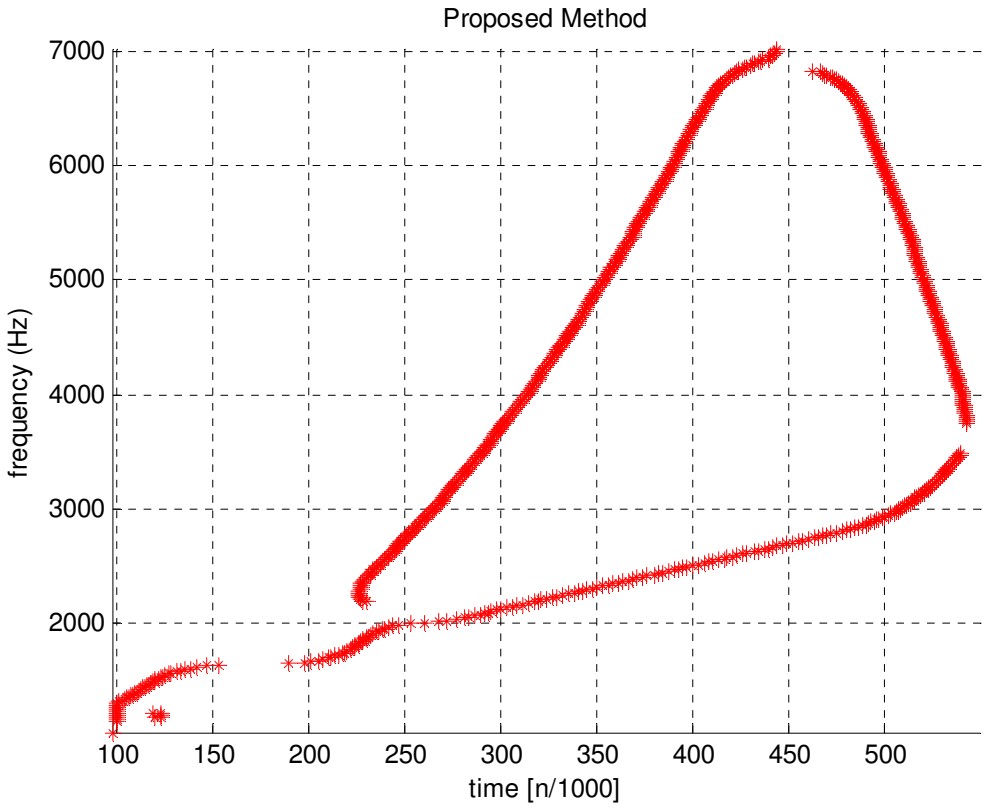

Figure 45: Results from Proposed Method with Peak Detection (Window Size = 8*Period of i, $\tau = 1$, Scanning Frequency Range i = 100-8000Hz with Interval of 10 Hz, Cosine-Sine Projection)

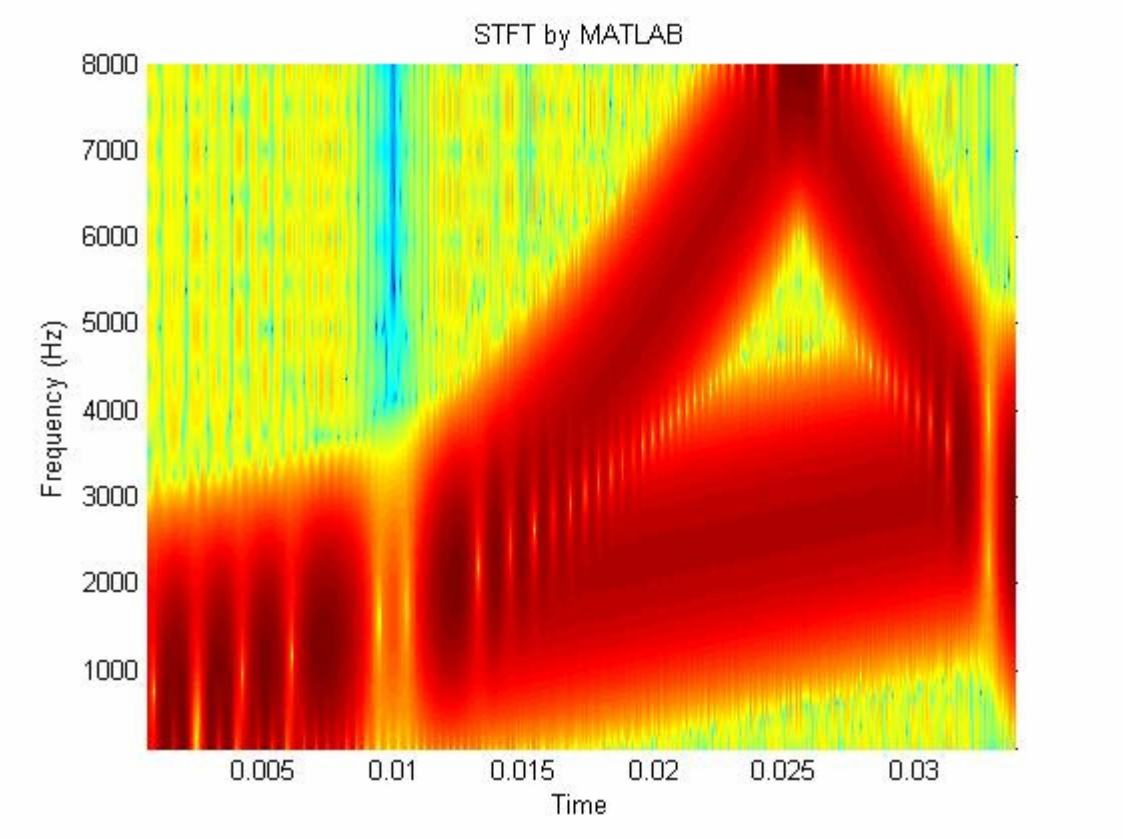

Figure 46: Results from STFT (Spectrogram) by MATLAB (Window Size = 16, FFT Size = 1024, 100% Overlapping)

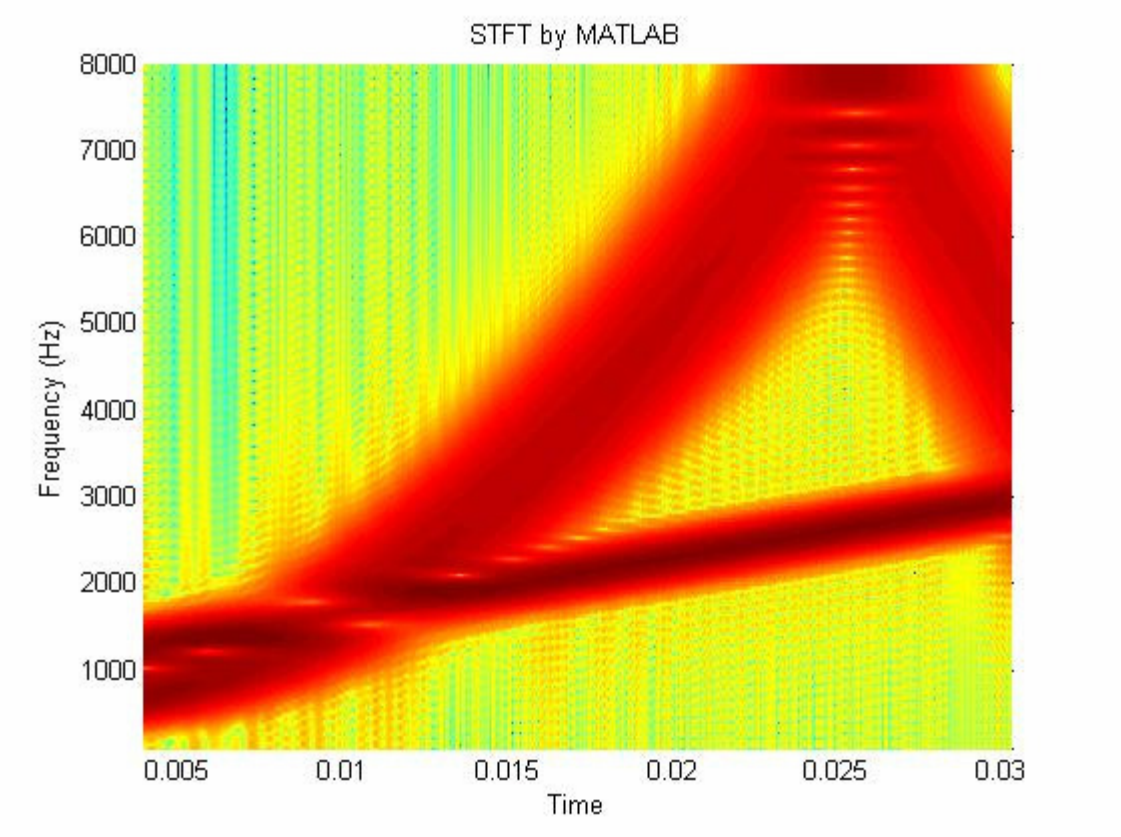

Figure 47: Results from STFT (Spectrogram) by MATLAB (Window Size = 128, FFT size = 1024, 100 % Overlapping)

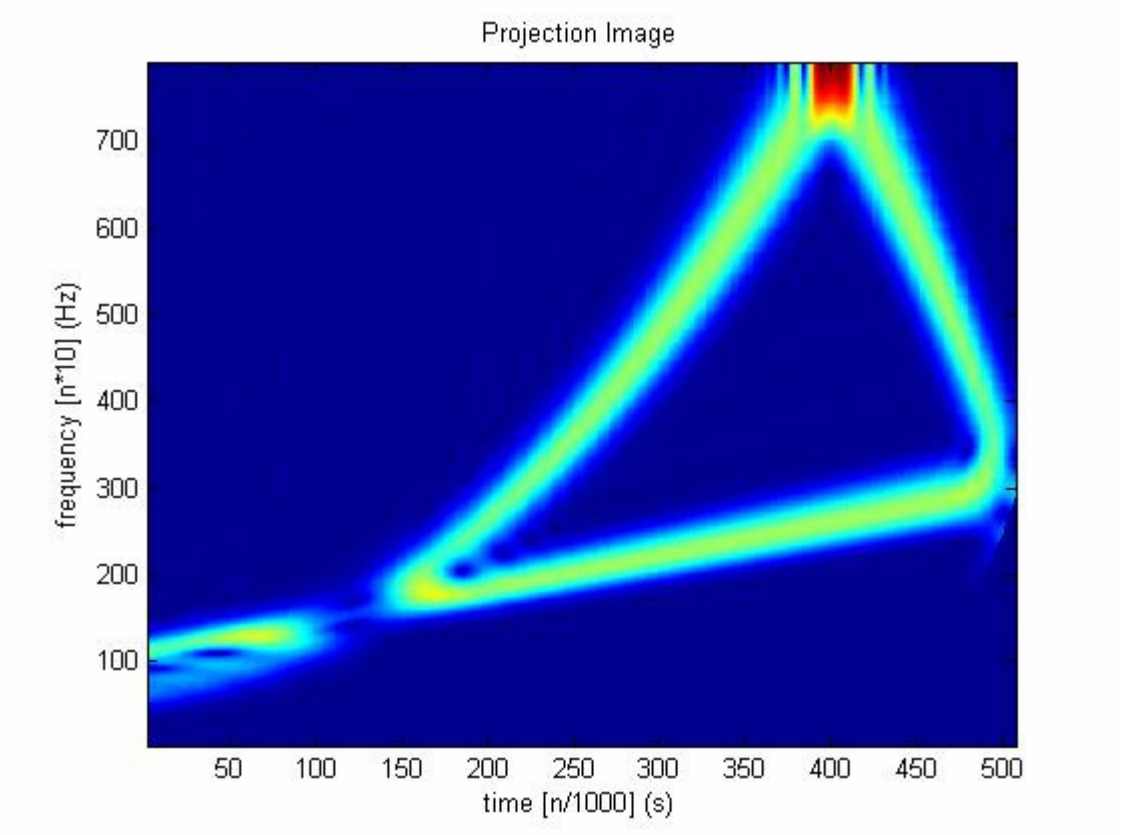

Figure 48: Results from Proposed Method before Peak Detection (Window Size = 8*Period of i, $\tau = 1$, Scanning Frequency Range i = 100-8000Hz with Interval of 10 Hz, Cosine-Sine Projection)

CHAPTER 4

REAL-TIME HARDWARE IMPLEMENTATION

Because of the adaptability of the algorithm, it can be tailored to fit on a flash drive of a Digital Signal Processing board and can produce meaningful result in real-time. In this paper, the Texas Instrument C6713 Digital Signal Processing (DSP) board is chosen for testing and analyzing generated signals which possess different spectral properties. The results are displayed in real-time to illustrate how the proposed algorithm performs in practical applications. This board was chosen because of its availability at the University of New Orleans and its popularity as a general Digital Signal Processing (DSP) board in many practical applications. The board has a moderate processing speed of about 200 MHz. However, with its memory space of 16MB and its sound processing unit, which has both stereo inputs and outputs, it serves the intended purpose of testing the versatility of the proposed algorithm.

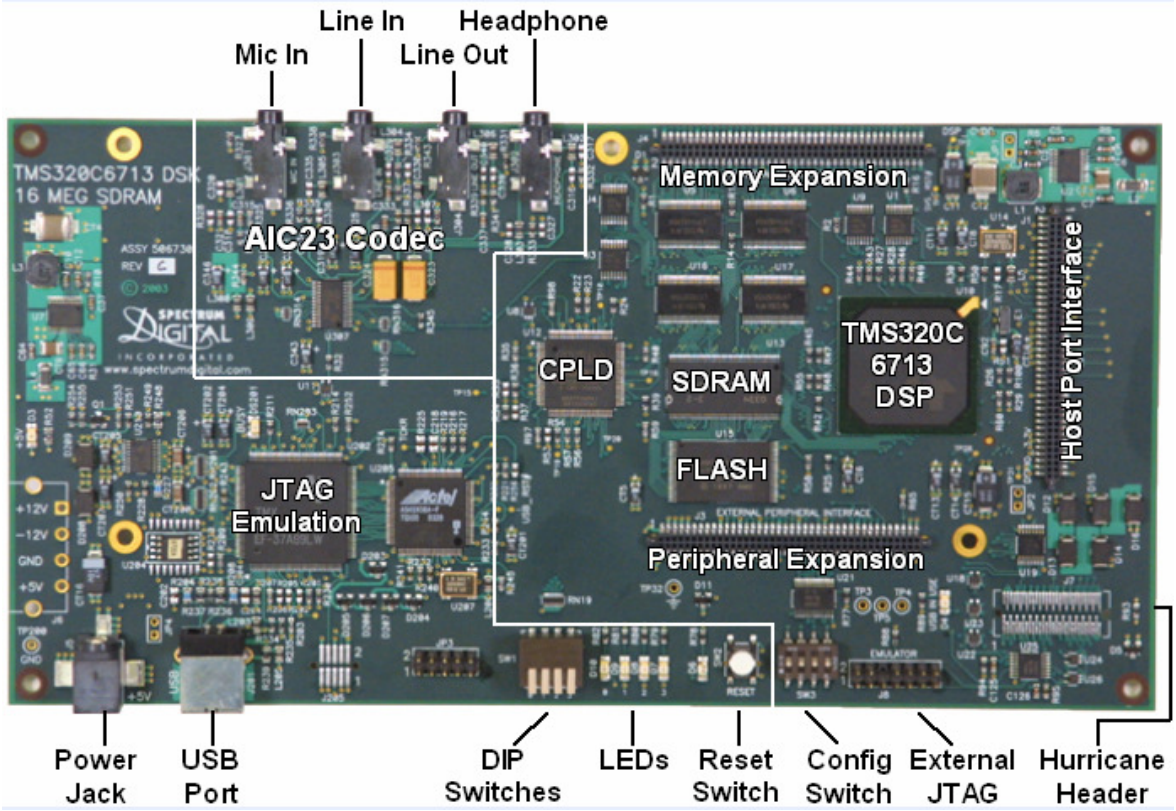

Figure 49: TMS320C6713 DSP Development Board
(Image by Texas Instruments, Inc.)

I. DSP Board Specifications:

- TMS320C6713 Board features [11]:
- 512K Flash and 16MB SRAM for application binary and memory space
- Code Composer Studio™ Integrated Development Environment software (somewhat like Visual Studio)
- USB communication to host computer
- +5V Power supply (with AC adapter)
- 24-bit stereo audio codec (with two input lines and two output lines)
- Digital Signal Processor features [11]:

- Eight 32-bit instructions/cycle
- 32/64- bit data word
- 225-, 200Mhz (GDP), and 200-, 167Mhz clock rates
- 4.4-, 5-, 6 Instruction cycle times
- 1800/1350, 1600/1200, and 1336/1000 MIPS/MFLOPS
- Two ALUs (fixed point)
- Four ALUs (floating and fixed point)
- 32-bit general purpose registers
- Instruction packing to reduce code size
- All instructions conditional

II. DSP code design and implementation

The Integrated Development Environment (IDE) software Code Composer Studio included in the DSP kit is used for implementing and designing a test application for the proposed algorithm. The IDE is a user-friendly interface that is similar to Visual Studio which eases the user in coding DSP algorithms. Most of the board's initialization and configuration are done with minimal user's efforts. For the test application, the code is developed in C++ language.

The board must be powered up and connect to the host computer which has Code Composer Studio installed before starting Code Composer Studio. After loading the project file (*.pjt), the user can select the main application code (*.c file) to begin coding.

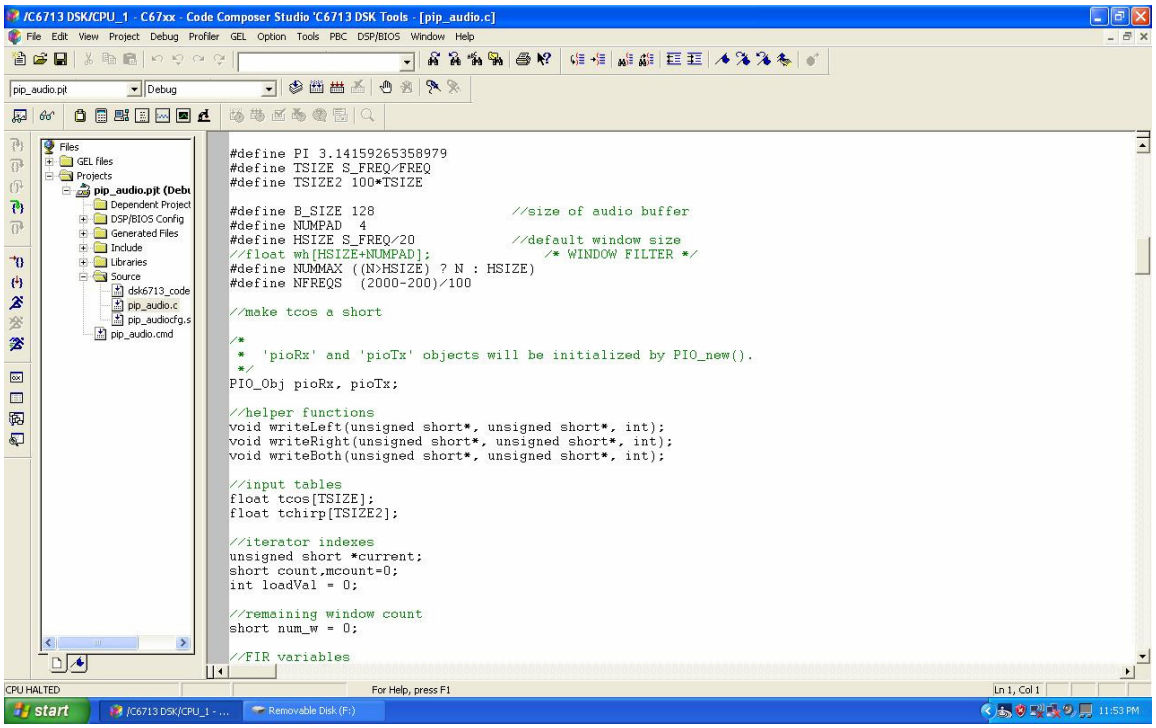

Figure 50: Code Composer Studio Interface

The sampling rate and buffer size for the test application were selected as 8 kHz and 128 samples, respectively. The application also is designed to use buffered pipe input/output in which a queue-type buffer is provided by the IDE for importing and outputting sound data from the outside world. Since buffer pipe is used, the applications files will have the names beginning with ***pip_audio***. Sampling rate and buffer size settings can be set by changing the properties of the file "***pip_audio.cdb***" and modifying the #define preprocessor.

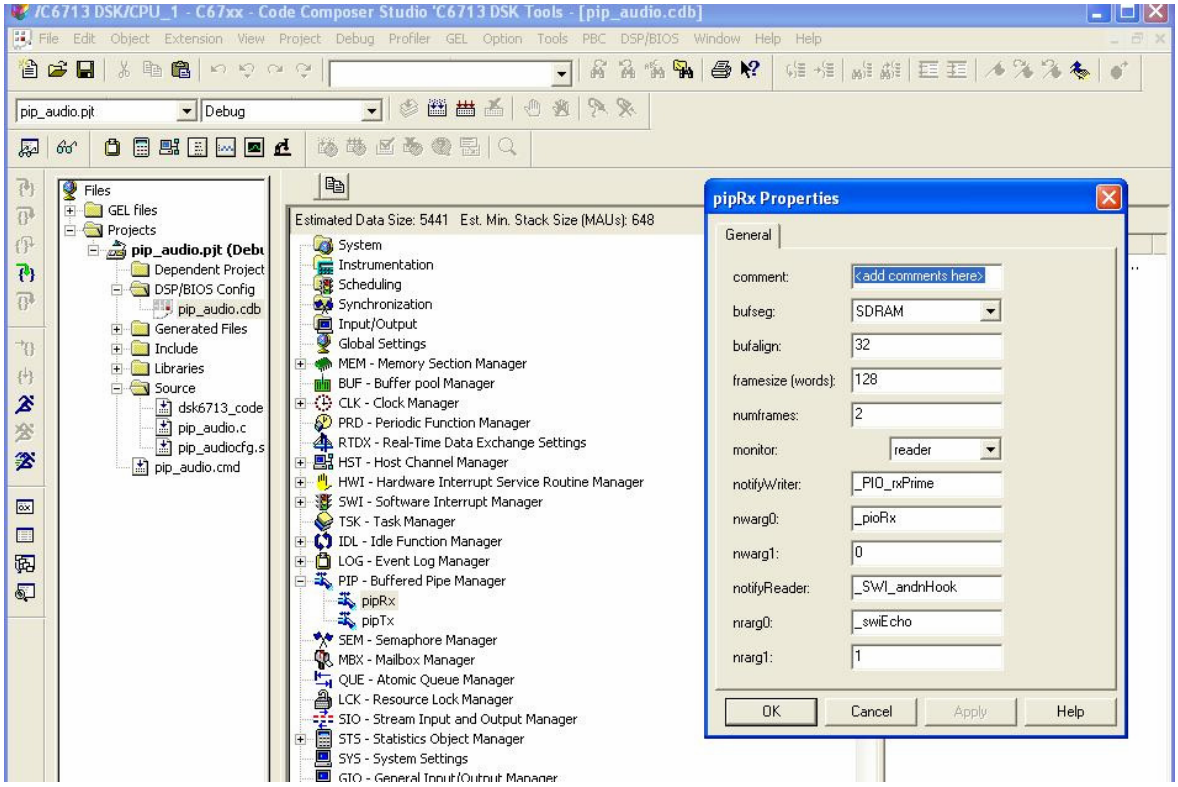

Figure 51: Setting the Number of Samples per Buffer Size

The sampling frequency can be set to 8 KHz by modifying the file "***pip_audio.c***," the main source file for this project.

```
#ifdef AIC23_REG8_DEFAULT

#undef AIC23_REG8_DEFAULT

#define AIC23_REG8_DEFAULT      AIC23_REG8_8KHZ
```

These lines must be placed after the "include" statements and before initializing at the input/output DSP sound. The sampling frequency can be changed to any value of 8, 32, 44, 48, and 96 KHz.

Since the input and output buffers and any filters coefficient used are floating-point variables and very large in size, these variables must be put in extended memory, SRAM, so that the remaining program code can fit on the flash drive space. The following steps need to be done in order to place those data arrays in extended memory.

Append the following lines at the end of the file ***pip_audio.cmd*** to define a new section in extended memory. In Code Composer Studio, the files that are ended with extension ***.cmd*** are called linker files. These files organize program code in memory and link various modules and/or libraries that the main program needs in order to run successfully.

```
SECTIONS

{

      .EXTRAM :> SRAM

}
```

In the main application file, ***pip_audio.cmd***, for each data array that needed to be in extended memory, the following statements are placed after the data is defined.

```
float out[B_SIZE] = {0.0};          // output buffer with size=B_SIZE

#pragma DATA_ALIGN(out, sizeof(float))

#pragma DATA_SECTION(out,".EXTRAM")
```

These statements align the data array on float-size boundary, which is necessary for some filtering operations and place the array on .EXTRAM as defined earlier.

The test application uses DSP/BIOS to schedule real-time processing and displaying the results. DSP/BIOS is a scalable real-time kernel. It provides preemptive multi-threading, hardware abstraction, and real-time analysis. A test signal is generated while the processing is being done for more efficient testing. To apply projection of coefficient segments on the test signal, a library Texas Instruments (TI) function, called ***DSPF_sp_dotprod***, is used. The name stands for single-precision dot product. TI provides a "hand made" library that contains a DSP function coded in Assembly language by design engineers for each particular DSP board to provide the users with the fastest and most efficient way of utilizing the DSP processors. In order to use this function, the library file "***dsp67x.lib***" is added to the project's environment and the header file, so that the function is included at the beginning of the main test application code:

```
#include <DSPF_sp_dotprod.h>
```

After including the header file and adding the library, the projection can be done by calling the ***DSPF_sp_dotprod*** function in the test application code. Real-time processing is done by using the software interrupt service function which is invoked every 3 millisecond interval.

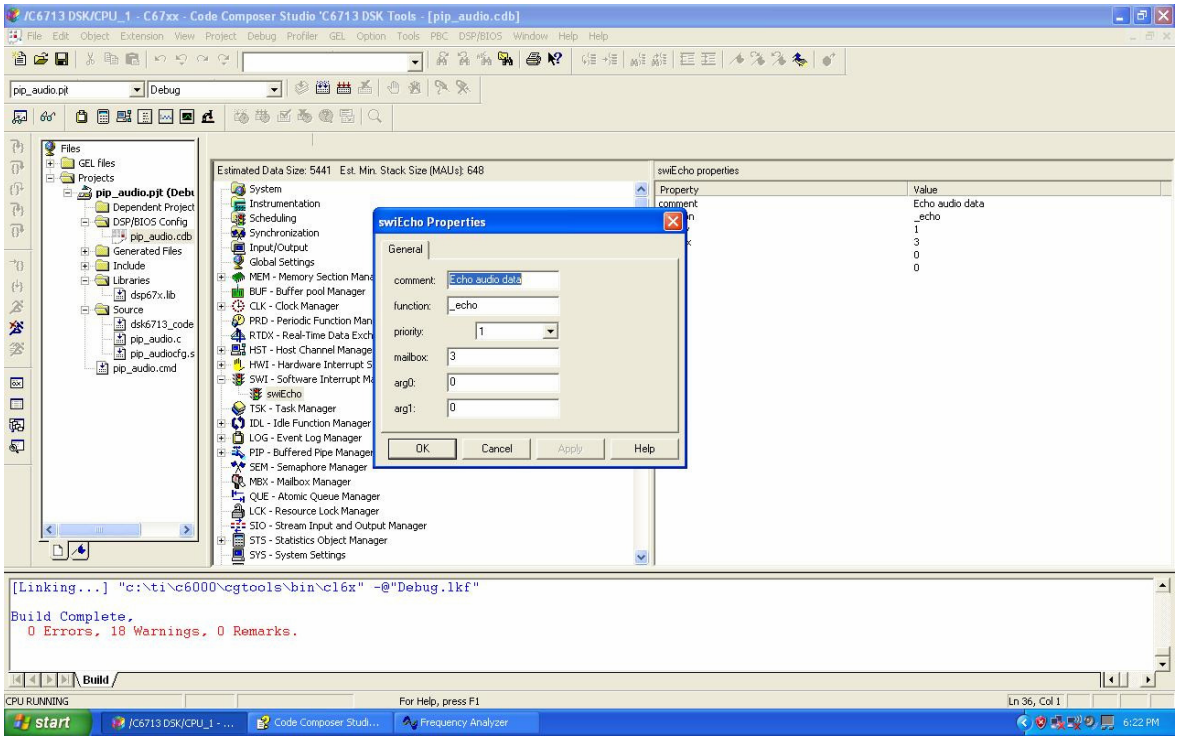

Figure 52: Configure Setting for Software Interrupt Intervals

The value set in the ***mailbox*** determines the time delay between software interrupt intervals. Each iteration corresponds to one millisecond assuming that the value in the property **CLK-Clock Manager** is set to 1000 microsecond intervals (default setting). In this case, mailbox's value is set to 3, or 3 millisecond intervals.

In order for fast projecting the coefficients on the signal, the coefficients are pre-generated by MATLAB in table arrangement and are loaded into memory. When the actual projection occurs, an index into the table is calculated and the appropriate coefficients are loaded one at a time. In the

test application, a header file (.h) is included at the beginning of the main source code. For example:

```
#include "ing.h"        //cosine kernel for frequency from 200-2000 Hz
```

The coefficients from the file ***ing.h*** are float-size aligned and are loaded into extended memory (SRAM) to save space for the program code.

```
#pragma DATA_ALIGN(ing, sizeof(float))

#pragma DATA_SECTION(ing,".EXTRAM")
```

c. Results and Analysis

Testing on the DSP board is done considering some modifications with respect to the proposed work presented earlier in this thesis so that instantaneous results are obtained. The inputs are generated by the DSP while processing takes place. The results are outputted into audio channel LINEOUT and displayed on computer monitor by the Frequency Analyzer v.2.0 software provided by Reliable Software (1996-2004).

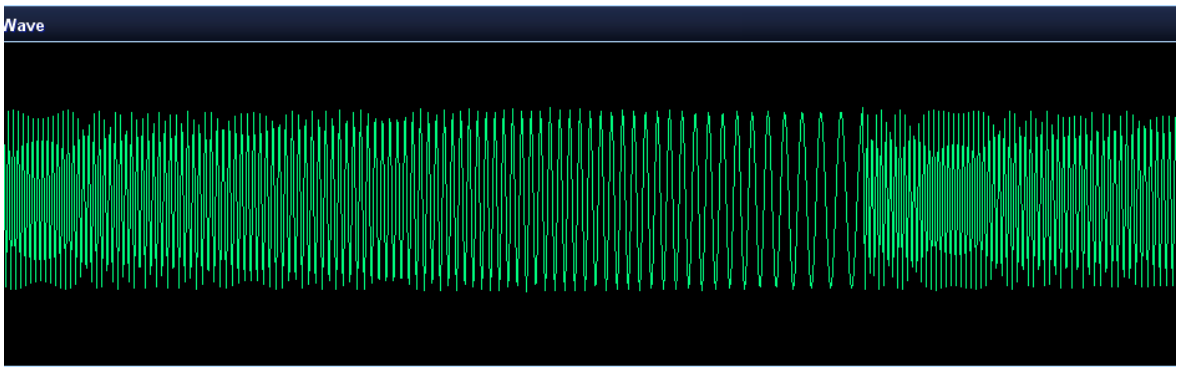

Figure 53: Input Signal for Hardware Testing (Frequency = 1000 Hz, Sampling Rate = 8000 Hz, Chirp-rate = 0.001)

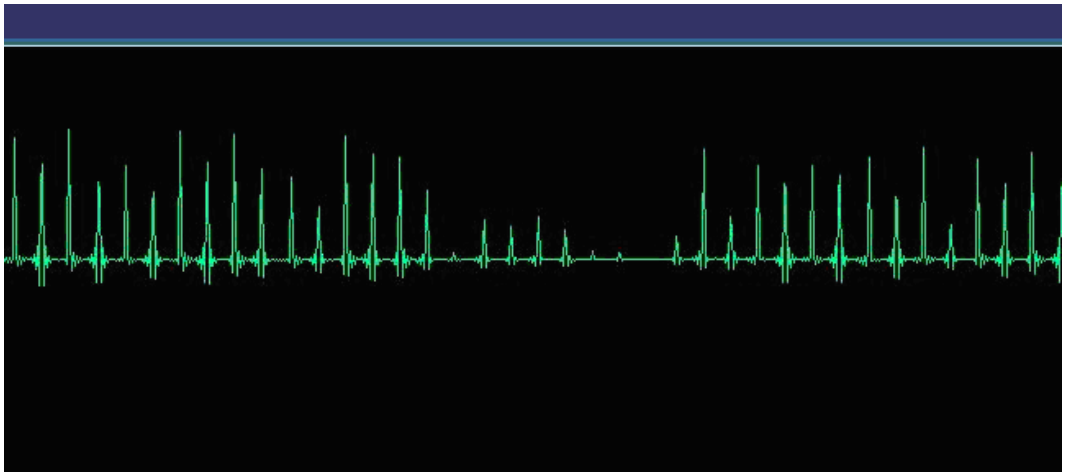

Figure 54: Output Signal by Hardware Testing (128 Samples/Buffer, $\tau = 9$, i = 200 to 2000 Hz with 100 Hz Interval, Cosine Only)

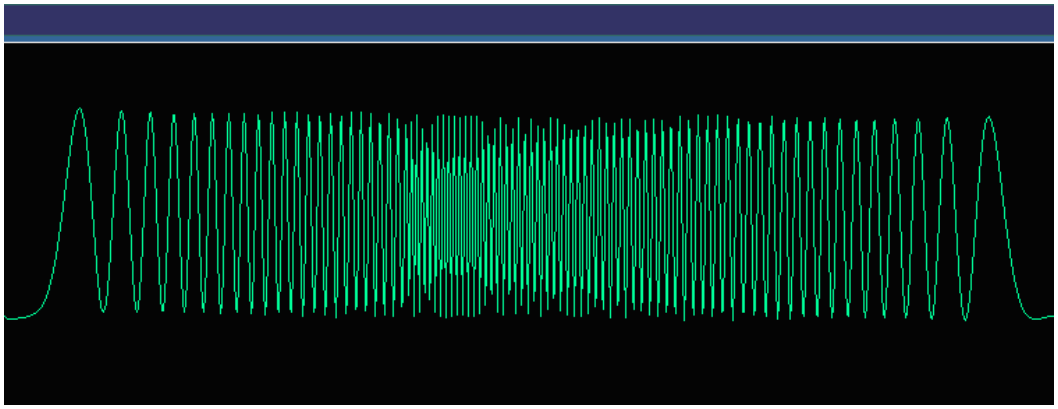

Figure 55: Input Signal for Hardware Testing (Frequency = 1000 Hz, Sampling Rate = 8000 Hz, Chirp-rate = 0.002)

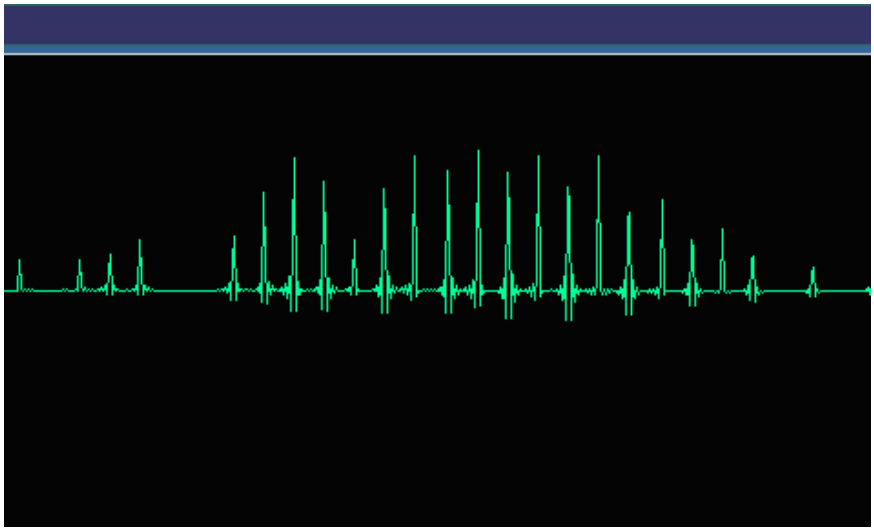

Figure 56: Output Signal by Hardware Testing (128 Samples/Buffer, $\tau = 9$, i = 200 to 2000 Hz with 100 Hz Interval, Cosine Only)

Analysis: Figure 53 is the input signal displayed in the time domain. The chirp is repeated whenever the highest frequency is reached. With the sampling rate set at 8000 Hz, the highest frequency component is at 4000 Hz, approximately. Figure 54 shows the real-time results obtained directly at the audio output channel of the DSP board. Each peak in the result is an indication of the

frequency found at that point in time. There are a number of errors which mostly due to the fact that the time-shift parameter τ is set at 9 samples and only cosine is used for projection. The overall result is consistent with the input chirp of 0.001. A virtual smooth envelop can be seen surrounding the peaks. The result of a higher input chirp of 0.002 in Figure 55 is also consistent: a narrower envelop is seen surrounding the peaks in Figure 56.

CHAPTER 5

CONCLUSIONS

Frequency estimation is the method of obtaining instantaneous spectral components of a signal. These spectral components can be used to analyze signals more thoroughly compared to traditional techniques of analyzing signals, such as the Fourier Transform approach. Instantaneous frequency estimation is very useful in a wide range of fields including communication, control systems, and radar imaging. Since embedded computerized systems, microcontrollers, and digital signal processors are extensively available as tools to analyze signal properties, it is necessary to design a fast and simple algorithm in the sense of its minimal complexity to run on these embedded computers. Not only the algorithm must have small complexity, but its performance must be acceptable to industrial and researching standards. Moreover, the algorithm's adaptability is also important in the sense that it should be scalable for different hardware speeds and resources. Recently, researchers have used several methods for frequency estimation such as STFT, Wavelet Transform, and Chirplet Transform, with STFT being the most popular due to its simplicity and small complexity in computation. However, the accuracy of STFT is directly related to its window size which can be hard to adapt to the dynamics of nature's spectral components. As the uncertainty principle states, high resolution in the time domain causes low resolution in the frequency domain and vice versa. For STFT to be accurate, as demonstrated in Chapter 3, an optimal window size must be found by iteratively searching through all possible window size to

find the best match. STFT also performs poorly for multi-component signals as pointed out in Chapter 3.

Chirplet Transform proves to be accurate and work well for multi-component signals. It involves projecting a family of chirplet functions, derived from a primitive function called mother chirplet, on a windowed signal to find the best chirplet function that matches the instantaneous frequency components. However, in order to obtain accurate matches, numerous parameter adjustments, and as a result numerous computations are required. Therefore, this transform is unrealizable for low speed embedded systems.

Our proposed method uses concepts similar to the Chirplet Transform. More specifically, a set of different properties of basis functions are projected onto the signal to find the best match. Our method is characterized by short adaptable window sizes and simple basis functions which match the properties of the signal within short time intervals. From the complexity point of view, our method introduces a complexity of $O(2n)$ per searched spectral component which leads to a complexity of $O(2nm)$ where m is the number of searchable spectral components. On the other hand, STFT requires a complexity of $O(nlogn)$ for each Fast Fourier Transform performed and an overall complexity of $O(mnlogn)$ if m different window sizes are used. Our method also proposes a set of adjustable parameters which adapt to different memory sizes and speed. As illustrated in Chapter 4, changing the parameter values in order to enable real-time processing still produces reasonable results.

BIBLIOGRAPHY

1. Boashash, B. Estimating and Interpreting the instantaneous frequency of a signal . Proc. IEEE, vol 80, no. 4 pp.520-568, Apr. 1992.
2. Steeghs, P. and Drijkoningen, G. *Time-frequency analysis of seismic reflection signals*. Proc. IEEE,ICASSP, vol.5, pp. 2972–2975, 1996.
3. J. E. Odegard, R. G. Baraniuk, and K. L. Oehler. *Instantaneous frequency estimation using the reassignment method*. Proc. Soc. Exploration Geophys. 67thAnnu. Meeting, Dallas, TX, Nov. 2–7, 1997.
4. P. Duvaut, A. Doucet, C. Veaux, and P. Flandrin. Instantaneous frequency estimation: Bayesian approaches versus reassignment—Application to gravitational waves. Proc. IEEE ICASSP, vol. 5, 1996.
5. Bultan, Aykut. *A Four-Parameter Atomic Decomposition of Chirplets*. IEEE Transactions on Signal Processing, vol 47, no. 3, March 1999.
6. Mann,Steve and Haykin, Simon. *The Chirplet Transform: Physical Considerations*. IEEE Transactions on Signal Processing, vol 43, no. 11, Nov 1995.
7. Angrisani, L. and D'Arco M. A Measurement Method Based on an Improved Version of the Chirplet Transform for Instantaneous Frequency Estimation. IEEE, 2001.
8. Barkat, B. and Boashash, B. "Instantaneous frequency estimation of the polynomial FM signals using the peak of the PWVD: Statistical performance in the presence of additive Gaussian noise,"IEEE Trans. Signal Processing, vol. 47, pp2480-2490, 1999.
9. Qian, S. and Chen, D., Adaptive Chirplet Based Signal Approximation, IEEE, 1998.
10. Qian, S. and Chen, D., *Joint Time-Frequency Analysis*. Prentice-Hall, 1996.
11. TMS320C6713 Floating-point Digital Signal Processor. Texas Instruments, Dec 2001.
12. Code Composer Studio Help Documentation. Texas Instruments, Inc., 2001.

APPENDIX A

MATLAB SIMULATION CODE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%test center chirp
c = [];
s_freq = 16000;
t = 0:1:550-1;
freq = 300;
stepc = 10;
c = 0.00001;
count = 0;
c2=[];
vchirp = [100:stepc:8000];
ind = [];
f = [];

ncomps = 2;     % number of frequency components
y=[];
imp = [];       % projection image
% for s=vphase
    c1=[];
    %t = 0:1/s_freq:10*1/freq;
    x = cos(2*pi*500*(1+0.00003*t.^2).*t/s_freq) +
cos(2*pi*1000*(1+0.002*t).*t/s_freq+pi/3);% +
cos(2*pi*(freq*2)*(1+c*t).*t/s_freq+pi/3);
    %x = cos(2*pi*freq*(1+c*t.^2).*t/s_freq);
    g1=gausswin(100,2);
    figure(1),plot(x),title('Input
Signal'),xlabel('Time[n/1000]'),ylabel('Amplitude')
    tic
    for i=vchirp
        %t = 0:1/s_freq:floor(1/freq);
        t = 0:1/s_freq:8/i;
        %g = exp(-[-length(t)/2+1:length(t)/2].^2/(2*(length(t)/4).^2));
        g = gausswin(length(t),2)';
        h1 = cos(2*pi*i.*t).*g;
        h2 = sin(2*pi*i.*t).*g;
        intv = 1;
        y1 = zeros(1,length(x));
        for j=1:intv:length(x)-length(h1)
            y1(j)=sum(x(j:j+length(h1)-1).*h1);
        end
```

```
        y2 = zeros(1,length(x));
        for j=1:intv:length(x)-length(h2)
            y2(j)=sum(x(j:j+length(h2)-1).*h2);
        end

        y = sqrt(y1.^2 + y2.^2)/sum(h1.^2 + h2.^2);
        imp = [imp y'];
        y = filter (g1,1,y)./sum(g1.^2);
        count = count + 1;

        %pause
        % thredshold
          ith = find (y < 0.3);
          y(ith) = 0;

        % Find local maximum
         imax = zeros(1,ncomps);
         i_imax = 1;
         for j=2:length(y)-1
             if (y(j) > y(j-1) && y(j) > y(j+1) && i_imax <= ncomps)
                imax(i_imax) = j;
               i_imax = i_imax+1;
             end
         end

        % Calculate slope for each line, solve collision
        % The hard way
%          if (length(ind) > 1)
%             for j=1:ncomps
%                 ix = (i-vchirp(0))/stepc;
%                 ax = repmat(ix,[1 ncomps]);
%                 mtmp = abs(repmat(imax(j),[ncomps 1])-ind(:,ix-
1+1))./(ix+1+1);
%                 iMaxtmp = imax(j);
%
%                 %swap
%                 imax(j) = imax(min(mtmp));
%                 imax(min(mtmp)) = iMaxtmp;
%             end
%          end

%         [m i] = max(y);
%         ind = [ind i];
        ind = [ind imax'];

        % figure(2),plot(h1),title('kernel');
        figure(3),plot(y),title('output');% set(gca, 'ylim',[0 1.5]);
%         title(num2str(i))
        %pause

    end
    toc
%figure(4),plot(ind,vchirp, 'r*'),title('chirp');
```

```
xLab = strcat('frequency [n*',num2str(stepc),'] (Hz)');
figure(6),imagesc(imp'),set(gca,'YDir','norm','XDir','norm'),title('Projectio
n Image'),ylabel(xLab),xlabel('time [n/1000] (s)')

%Plotting
figure(4),set(gca, 'xlim',[0 length(x)]),ylabel('frequency
(Hz)'),xlabel('time [n/1000]'),title('Proposed Method');
grid on
hold on
for i=1:ncomps
    figure(4),plot(ind(i,:),vchirp,'r*')
end

% % plot real chirp traces
% plot(1000*(1+0.002.*t));
% plot(500*(1+0.00003*t.^2));
% hold off

% Image plot for projecting


%STFT by MATLAB spectrogram
figure(5),spectrogram(x,16,15,1024,s_freq,'yaxis'),title('STFT by
MATLAB'),set(gca,'YLim',[100 8000])
```

APPENDIX B

Matlab Scripts for Generating DSP Coefficients

```
clear all
close all
% Generate cosine kernels for frequency from 200-2000Hz
c = [];
s_freq = 8000;
t = 0:1:128-1;
freq = 3800;
stepi=100;
stepphase=18/360;
count = 0;
c2=[];

index = 0;      % loop index
vgau = [];      % the Gaussian vector
vlen = [];      % the lengths vector
vinw = [];      % the windowed input
sumh = [];      % sum of kernels

vfreq = [200:stepi:2000];
vphase = [0:stepphase:1-stepphase];
% for s=vphase
    c1=[];
    for i=vfreq
        t = 0:1:floor(3*s_freq/i);
        %g=exp(-[-length(t)/2+1:length(t)/2].^2/(2*(length(t)/4).^2));
        g = gausswin(length(t),2)';
        h1 = cos(2*pi*(t)*i/s_freq).*g;

        %Reverse h1
%         n = length(t) - t;
%         h1 = h1(n);
        vinw = [vinw h1];
        %vgau = [vgau g];
        vlen = [vlen length(t)];
        sumh = [sumh sum(cos(2*pi*(t)*i/s_freq).^2)];
%         count = count + 1;
    end
%     c2=[c2;c1];
% end
count
% c=max(c2,[],1);
% p=max(c2,[],2);
```

```
% figure(1),imagesc(vfreq,vphase,c2)
%figure(2),plot(vphase,p),title('phase around the center')
% ind = find (c1 < 0.1);
% c1(ind) = 0;
%figure(3),plot(vfreq,c1),title('sums of frequencies around the center')

% write to file
cd l:\Freq' Est'\
fid = fopen('ing.h','w');

% Write windowed inputs to be compared to file
fprintf(fid, '#define NG ');
fprintf(fid, '%d',length(vinw));
fprintf(fid, ' // filter array length\n');
fprintf(fid, 'float ing[');
fprintf(fid, '%d', length(vinw));
fprintf(fid, '] = {');
fprintf(fid, '%f, ',vinw(1:length(vinw)-1));
fprintf(fid, '%f', vinw(end));
fprintf(fid,'};\n');

% Write lengths of inputs to file
fprintf(fid, '#define NLENWG ');
fprintf(fid, '%d',length(vlen));
fprintf(fid, ' // lengths of inputs\n');
fprintf(fid, 'short inglen[');
fprintf(fid, '%d', length(vlen));
fprintf(fid, '] = {');
fprintf(fid, '%d, ',vlen(1:length(vlen)-1));
fprintf(fid, '%d', vlen(end));
fprintf(fid,'};\n');

% Write sum of squared kernels to file
fprintf(fid, '#define NSUMH ');
fprintf(fid, '%d',length(sumh));
fprintf(fid, ' // kernel sum array length\n');
fprintf(fid, 'float sumh[');
fprintf(fid, '%d', length(sumh));
fprintf(fid, '] = {');
fprintf(fid, '%f, ',sumh(1:length(sumh)-1));
fprintf(fid, '%f', sumh(end));
fprintf(fid,'};\n');

fclose(fid);

clear all
close all
% Generate sine kernels for frequency from 200-2000Hz
c = [];
s_freq = 8000;
t = 0:1:128-1;
freq = 3800;
stepi=100;
stepphase=18/360;
count = 0;
```

```
c2=[];

index = 0;      % loop index
vgau = [];      % the Gaussian vector
vlen = [];      % the lengths vector
vinw = [];      % the windowed input
sumh = [];      % sum of kernels

vfreq = [200:stepi:2000];
vphase = [0:stepphase:1-stepphase];
% for s=vphase
    c1=[];
    for i=vfreq
        t = 0:1:floor(3*s_freq/i);
        %g=exp(-[-length(t)/2+1:length(t)/2].^2/(2*(length(t)/4).^2));
        g = gausswin(length(t),2)';
        h1 = sin(2*pi*(t)*i/s_freq).*g;

        %Reverse h1
%         n = length(t) - t;
%         h1 = h1(n);
        vinw = [vinw h1];
        %vgau = [vgau g];
        vlen = [vlen length(t)];
        sumh = [sumh sum(cos(2*pi*(t)*i/s_freq).^2)];
%         count = count + 1;
    end
%     c2=[c2;c1];
% end
count
% c=max(c2,[],1);
% p=max(c2,[],2);
% figure(1),imagesc(vfreq,vphase,c2)
%figure(2),plot(vphase,p),title('phase around the center')
% ind = find (c1 < 0.1);
% c1(ind) = 0;
%figure(3),plot(vfreq,c1),title('sums of frequencies around the center')

% write to file
cd l:\Freq' Est'\
fid = fopen('sing.h','w');

% Write windowed inputs to be compared to file
fprintf(fid, '#define SNG ');
fprintf(fid, '%d',length(vinw));
fprintf(fid, ' // filter array length\n');
fprintf(fid, 'float sing[');
fprintf(fid, '%d', length(vinw));
fprintf(fid, '] = {');
fprintf(fid, '%f, ',vinw(1:length(vinw)-1));
fprintf(fid, '%f', vinw(end));
fprintf(fid,'};\n');

% Write lengths of inputs to file
```

```
fprintf(fid, '#define SNLENWG ');
fprintf(fid, '%d',length(vlen));
fprintf(fid, ' // lengths of inputs\n');
fprintf(fid, 'short singlen[');
fprintf(fid, '%d', length(vlen));
fprintf(fid, '] = {');
fprintf(fid, '%d, ',vlen(1:length(vlen)-1));
fprintf(fid, '%d', vlen(end));
fprintf(fid,'};\n');

% Write sum of squared kernels to file
fprintf(fid, '#define SNSUMH ');
fprintf(fid, '%d',length(sumh));
fprintf(fid, ' // kernel sum array length\n');
fprintf(fid, 'float ssumh[');
fprintf(fid, '%d', length(sumh));
fprintf(fid, '] = {');
fprintf(fid, '%f, ',sumh(1:length(sumh)-1));
fprintf(fid, '%f', sumh(end));
fprintf(fid,'};\n');

fclose(fid);
```

APPENDIX C

DSP SIMULATION CODE

```
/*
 *  Copyright 2003 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/* "@(#) DSP/BIOS 4.90.270 01-08-04 (bios,dsk6713-c04)" */
/*
 *  ======== pip_audio.c ========
 *
 *  This example demonstrates the use of IOM drivers with PIPs using
 *  the PIO adapter with a user defined device mini-driver called
 *  "udevCodec". The application performs a loopback.  That is, audio data is
 *  read from one PIP connected to an input IOM channel, and the data is
 *  written back out on a PIP connected to an output IOM channel.
 *
 *  The following objects need to be created in the DSP/BIOS
 *  configuration for this application:
 *
 *  * A UDEV object, which links in a user device driver. In this case
 *    the UDEV is a codec based IOM device driver.
 *
 *  * A SWI object named swiEcho. Configure the function as _echo,
 *    and the mailbox value as 3.
 *
 *  * 2 PIP objects, one named pipTx, the other pipRx. The length of the
 *    buffers should be the same and can be any size. See the comments
 *    by the declarations below of pipTx and pipRx for the writer and
 *    reader notify function settings.
 *
 *  * A LOG object named trace, used for status and debug output. Can be
 *    any size and can be circular or fixed.
 */
```

```
#include <std.h>

#include <log.h>
#include <pip.h>
#include <swi.h>
#include <sys.h>

#include <iom.h>
#include <pio.h>
#include <math.h>
                                                        /*VERY IMPORTANT*/
#include "coeffs.h"                                              //filter coefficients
#include "ing.h"
                                                //cosine kernel for frequency from 200-2000 Hz
#include "sing.h"      //sine kernel for frequency from 200-2000 Hz

//#include "sum_ing.h"    // sum of input <dot> gaussian (for chirp detect)
//#include "chirp.h"
#include <DSPF_sp_fir_gen.h>      //TI filter
//#include <DSPF_sp_fir_r2.h>
#include <DSPF_sp_dotprod.h> //dot product
#include <DSPF_sp_maxval.h>
#include <DSPF_sp_maxidx.h>

#ifdef _6x_
extern far LOG_Obj trace;
extern far PIP_Obj pipRx;
extern far PIP_Obj pipTx;
extern far SWI_Obj swiEcho;
#else
extern LOG_Obj trace;
extern PIP_Obj pipRx;
extern PIP_Obj pipTx;
extern SWI_Obj swiEcho;
#endif

//set sampling rate
#ifdef AIC23_REG8_DEFAULT
#undef AIC23_REG8_DEFAULT
#define AIC23_REG8_DEFAULT AIC23_REG8_8KHZ
#endif

#define S_FREQ  8000
#define BUFFSIZE 1

// Input Frequency
#define FREQ 1000
```

```
// Chirp Rate
#define C_RATE 0.001

//Output Amp
#define AMP 500

#define PI 3.14159265358979
#define TSIZE S_FREQ/FREQ
#define TSIZE2 100*TSIZE

#define B_SIZE 128                //size of audio buffer
#define NUMPAD  4
#define HSIZE S_FREQ/20
//default window size
//float wh[HSIZE+NUMPAD];
/* WINDOW FILTER */
#define NUMMAX ((N>HSIZE) ? N : HSIZE)
#define NFREQS  (2000-200)/100

//make tcos a short

/*
 *  'pioRx' and 'pioTx' objects will be initialized by PIO_new().
 */
PIO_Obj pioRx, pioTx;

//helper functions
void writeLeft(unsigned short*, unsigned short*, int);
void writeRight(unsigned short*, unsigned short*, int);
void writeBoth(unsigned short*, unsigned short*, int);

//input tables
float tcos[TSIZE];
float tchirp[TSIZE2];

//iterator indexes
unsigned short *current;
short count,mcount=0;
int loadVal = 0;

//remaining window count
short num_w = 0;

//FIR variables
short freq,wsize = 0;                                                    //
current window frequency and window size
```

```
float cRate = C_RATE;                                                    //
chirp rate (for testing)

float wh[B_SIZE]={0.0};
                                                                  /* WINDOW FILTER */
float out[B_SIZE]={0.0};
//float in[B_SIZE+NUMMAX-1+NUMPAD]={0.0};
//float ins[B_SIZE+NUMMAX-1]={0.0};
                                                                /* scratch buffer for input*/
//float whs[B_SIZE]={0.0};
                                                                                /*
scratch buffer for window */
short output[B_SIZE]={0};
//float ws[S_FREQ/20] = {0.0};

float s_in[B_SIZE*3] = {0.0};                                                    /*
scratch buffer for input*/
short aChirpBuf[B_SIZE] = {0};                                                   /*
chirp buffer */

// Align variables to boundary

#pragma DATA_ALIGN(s_in, sizeof(float))
//#pragma DATA_ALIGN(input, sizeof(float))
#pragma DATA_ALIGN(h, sizeof(float))
#pragma DATA_ALIGN(wh, sizeof(float))
#pragma DATA_ALIGN(ing, sizeof(float))
#pragma DATA_ALIGN(inglen, sizeof(short))
#pragma DATA_ALIGN(sumh, sizeof(float))
//#pragma DATA_ALIGN(sum_ing, sizeof(float))
#pragma DATA_ALIGN(out, sizeof(float))


// Force to external RAM

#pragma DATA_SECTION(ing,".EXTRAM")
#pragma DATA_SECTION(inglen,".EXTRAM")
#pragma DATA_SECTION(sing,".EXTRAM")
#pragma DATA_SECTION(singlen,".EXTRAM")
//#pragma DATA_SECTION(sum_ing,".EXTRAM")
//#pragma DATA_SECTION(ins,".EXTRAM")
#pragma DATA_SECTION(h,".EXTRAM")
#pragma DATA_SECTION(out,".EXTRAM")
#pragma DATA_SECTION(wh,".EXTRAM")
//#pragma DATA_SECTION(in,".EXTRAM")
#pragma DATA_SECTION(tcos,".EXTRAM")
#pragma DATA_SECTION(tchirp,".EXTRAM")
```

```
//#pragma DATA_SECTION(whs,".EXTRAM")
#pragma DATA_SECTION(s_in,".EXTRAM")
#pragma DATA_SECTION(sumh,".EXTRAM")
#pragma DATA_SECTION(output,".EXTRAM")


/*
 *  ======== main ========
 *
 *  Application startup funtion called by DSP/BIOS. Initialize the
 *  PIO adapter then return back into DSP/BIOS.
 */
main()
{
// init buffer
short i;
float tphase = 0.5;
freq = FREQ;
cRate = C_RATE;

count = 0;
//count=N-1;

LOG_printf(&trace,"Start processing\n");

//init input buffer
for (i=0; i < B_SIZE*2+4; i++)
s_in[i] = 0.0;

// make a cosine table
for (i=0; i < TSIZE; i++)
{
//tcos[i] =  cos (2*PI*(120.0/TSIZE)*(1+i*0.0001)*(float)i);
//tcos[i] =  cos (2*PI*FREQ*i/S_FREQ + PI/180) +(2*PI*FREQ*i/S_FREQ +
45*PI/180)+(2*PI*FREQ*i/S_FREQ + 90*PI/180);//+ cos (2*PI*900*i/S_FREQ);
tcos[i] =  cos (2*PI*freq*i/S_FREQ + 2*PI*2/3);
}


// make chirp table
for (i=0; i < TSIZE2; i++)
tchirp[i] = cos (2*PI*freq*(2-cRate*i)*i/S_FREQ);

/* For window filter*/
//wsize = S_FREQ/freq;

//for (i=0; i < wsize; i++)
```

```
//wh[i] =  cos (2*PI*freq*i/S_FREQ);

//a chirp window
wsize = TSIZE2;
cRate = cRate - 0.1;
for (i=0; i < B_SIZE; i++)
wh[i] = cos (2*PI*FREQ*(1+cRate*i)*i/S_FREQ);

wh[wsize]=0.0;
wh[wsize+1]=0.0;
wh[wsize+2]=0.0;
wh[wsize+3]=0.0;

memset(&s_in[0], 0, B_SIZE*3);

// Generate Gaussian windows use for filtering


    /*
     * Initialize PIO module
     */
    PIO_init();

    /* Bind the PIPs to the channels using the PIO class drivers */
    PIO_new(&pioRx, &pipRx, "/udevCodec", IOM_INPUT, NULL);
    PIO_new(&pioTx, &pipTx, "/udevCodec", IOM_OUTPUT, NULL);

    /*
     * Prime the transmit side with buffers of silence.
     * The transmitter should be started before the receiver.
     * This results in input-to-output latency being one full
     * buffer period if the pipes is configured for 2 frames.
     */
    PIO_txStart(&pioTx, PIP_getWriterNumFrames(&pipTx), 0);

    /* Prime the receive side with empty buffers to be filled. */
    PIO_rxStart(&pioRx, PIP_getWriterNumFrames(&pipRx));

    LOG_printf(&trace, "pip_audio started");
}


/*
 *  ======== echo ========
 *
 *  This function is called by the swiEcho DSP/BIOS SWI thread created
 *  statically with the DSP/BIOS configuration tool. The PIO adapter
```

```
 *  posts the swi when an the input PIP has a buffer of data and the
 *  output PIP has an empty buffer to put new data into. This function
 *  copies from the input PIP to the output PIP. You could easily
 *  replace the copy function with a signal processing algorithm.
 */
Void echo(Void)
{
short size,i,j,k,l,tlen,c_freq,index, mindex, ichirp, fintv = 0;
signed short signTh = 1;
float c_chirp, c_theta, maxSum, tempSum, cTempSum, sTempSum, s1, s2, smid, sleft, sright,left,mid,
right, tright, tleft, intv;
Bool done = FALSE;
   unsigned short *src, *dst;

//Int len = 1024;
//  unsigned short buf;

//src2=(unsigned short *)malloc(1024*sizeof(unsigned short));
   /*
    * Check that the precondions are met, that is pipRx has a buffer of
    * data and pipTx has a free buffer.
    */
   if (PIP_getReaderNumFrames(&pipRx) <= 0) {
      LOG_error("echo: No reader frame!", 0);
      return;
   }
   if (PIP_getWriterNumFrames(&pipTx) <= 0) {
      LOG_error("echo: No writer frame!", 0);
      return;
   }

   /* get the full buffer from the receive PIP */
   PIP_get(&pipRx);
   src = PIP_getReaderAddr(&pipRx);
   size = PIP_getReaderSize(&pipRx) * sizeof(short);

   /* get the empty buffer from the transmit PIP */
   PIP_alloc(&pipTx);
   dst = PIP_getWriterAddr(&pipTx);


/* To generate a chirp */
for (i=0; i < B_SIZE; i++)
{
s_in[i] = s_in[i+ B_SIZE];
s_in[i+B_SIZE] = s_in[i+B_SIZE*2];                    //copy previous and next segment
                                                                  //for filtering purpose
```

```
s_in[i+B_SIZE*2]= tchirp[count];                    //padded input for filtering
++count ;
if (count >= TSIZE2) count = 0;
}

// find center frequency
left = 200;
right = 2000;
intv = 100;                                        //interval = 100 Hz
done = FALSE;


/* Find the center frequency peak using linear search */
// i is the frequency index
// j is the compared freq length index

tlen = 10;
memset(&aChirpBuf[0], 0, B_SIZE);                  // clear chirp buffer
mindex = 0;                                                          // the
index of the peak
ichirp = 0;                                        // chirp buffer index
fintv = floor((right-left)/intv);                                     //
interval between freqs

for (i=left, j=0; i <= right; i=i+intv,j++)
{
tempSum = 0.0;
maxSum = 0.0;
for (k=0 ; k < B_SIZE; k+= tlen)
{
cTempSum = DSPF_sp_dotprod (&s_in[B_SIZE+k], &ing[index],inglen[j]);
//sTempSum = DSPF_sp_dotprod (&s_in[B_SIZE+k], &sing[index],inglen[j]);
tempSum = pow(cTempSum,2);// + pow(sTempSum,2);
//tempSum = 1;

if (tempSum > maxSum)
{
mindex=k;
maxSum = tempSum;
}
}

ichirp += fintv;

// construct a line
//for (k = 1; k < fintv; k++)
```

```
//
aChirpBuf[ichirp-fintv+k] = (mindex - aChirpBuf[ichirp-fintv])/fintv*k + aChirpBuf[ichirp-fintv];

aChirpBuf[ichirp] = mindex;                                          // index of this peak

index += inglen[j];
}

// The center frequency
//c_freq = (DSPF_sp_maxidx(&aMaxSum[0], NFREQS)+1)*intv+intv;


//generate chirp
for (i=0; i < B_SIZE; i++)
output[i] = 10*aChirpBuf[i];
//output[i] = (short)(1000*s_in[B_SIZE+i]);


//Normalize
//for (i=0; i<B_SIZE; i++)
//output[i] = (short)(1000*s_in[B_SIZE+i]);

writeBoth(output,dst,size);

   /* Record the amount of actual data being sent */
   PIP_setWriterSize(&pipTx, PIP_getReaderSize(&pipRx));

   /* Free the receive buffer, put the transmit buffer */
   PIP_put(&pipTx);
   PIP_free(&pipRx);
}

/* Output on the left channel*/
void writeLeft(unsigned short *src, unsigned short *dst, int size){
short i;
for (i = 0; i < size; i++) {
        if (i % 2 == 1 )
        {
           *dst = *src;
           src++;
        }
        else
            *dst = 0;
        dst++;
}
}
```

```
/* Output on the right channel*/
void writeRight(unsigned short *src, unsigned short *dst, int size){
short i;
for (i = 0; i < size; i++) {
        if (i % 2 == 0 )
        {
        *dst = (*src);
        src++;
        }
        else
            *dst = 0;
        dst++;
}
}

/* Output on both channels */
void writeBoth(unsigned short *src, unsigned short *dst, int size){
short i,j;
for (i=0, j=0; i < size; i++,j=j+2) {
        dst[j] = *src;
        dst[j-1] = *src;
        src++;
}
}
```

VITA

Cuong Mai was born in Rach Gia, Vietnam. He came to the United States in 1994 and received his first Bachelor of Science in Electrical Engineering at the University of New Orleans in 2003. He is currently working as a Software Engineer at Input/Output Inc. in New Orleans, Louisiana.