Computing the Cross Ambiguity Function – A Review

by

Christopher L. Yatrakis

Bachelor of Science in Computer Engineering
Binghamton University, State University of New York, 2001

THESIS

Submitted in partial fulfillment of the requirements for
the degree of

Master of Science of Electrical Engineering
Binghamton University, State University of New York
2005

Accepted in partial fulfillment of the requirements for the
degree of Master of Science in Electrical Engineering
in the Graduate School of
Binghamton University, State University of New York

2005

Mark L. Fowler ______________________________ January 28, 2005
Department of Electrical and Computer Engineering

Eva Wu ______________________________ January 28, 2005
Department of Electrical and Computer Engineering

Xiaohua Li ______________________________ January 28, 2005
Department of Electrical and Computer Engineering

Abstract

Computing the cross ambiguity function is essential for TDOA/DD emitter location. Many algorithms for computing the cross ambiguity function have been presented in literature throughout the years. However, the existing literature is lacking in providing analysis in how these algorithms apply to emitter location, in determining which methods are better under certain scenarios. There has also been no attempt to discuss the similarities and differences of multiple algorithms or provide an outline of the relationships of some of these algorithms. Some of these algorithms are very similar to each other and include techniques to reduce the computational load needed to accurately compute the ambiguity function. Four such algorithms, the "Fine-Mode", "Fine-Mode" Generic Filter, "Fine-Mode" Generic Filter Frequency Domain, and Two-Dimensional Cross Spectra, using windowing/filtering and decimation to approximate the ambiguity function that allow for accurate computation, as well as a "brute force" method are discussed and analyzed to provide similarities and differences in their structures. Relationships among the five methods are established and results from testing the five algorithms in an emitter location scenario to determine accuracy and computational complexity with varying sampling rates, decimation, and Doppler shift are given. The results show that there is no one clear best overall method that gives the best accuracy and best computational complexity. Given a particular frequency, sampling rate, and amount of decimation the best method in terms of accuracy and computational complexity varies. The two-dimensional Cross Spectra method does produce the worst TDOA accuracy among all the methods. Though the findings are promising, further study is needed for all the methods. Analysis of the results indicates that sources of error deal with curve-fit errors that can be attributed to the simulation software used and aliasing errors from the amount of decimation. Suggestions for further study are made and may be used to provide a roadmap for further work in this area. An objective of this thesis to consolidate into one single place the information on cross ambiguity function algorithms for five algorithms found in literature and their relationships between each of them was obtained.

Table of Contents

Table of Tables

Table of Figures

Chapter 1 Introduction

Multiple algorithms have been developed in order to compute the narrow-band cross-ambiguity function (CAF). These algorithms share many similarities in their structure (decimation, filtering, etc…). One application where these existing algorithms are applied is in emitter location. Emitter location uses techniques in the disciplines of estimation theory and signal processing to find transmitters of a tasked frequency and distance from a reference point of platforms (receivers) that collect data in order to use time-difference-of-arrival (TDOA or time-delay) and/or frequency-difference-of-arrival (FDOA or Doppler) to locate the precise location to a desired accuracy. In emitter location, the object is to compute the cross-ambiguity function as quickly as possible and use it to determine the most accurate position of the emitter. In existing literature, no detailed analysis has ever been performed that combines analysis of multiple CAF methods that highlights their similarities but at the same time compares and contrasts these algorithms for emitter location to provide insight on what algorithm to use, under certain conditions.

In order to provide this information, a detailed analysis on five cross-ambiguity algorithms was performed. The methods used for the analysis were the Filter Bank, "Fine-Mode", "Fine-Mode" Generic Filter, "Fine-Mode" Generic Filter Frequency Domain, and the Two-Dimensional Cross Spectra methods. This analysis included using mathematical derivations from literature for all the methods in order to indicate similarities and differences between the structures of the algorithms. Only ***existing*** algorithms were used for this analysis. ***No new algorithms*** were created and used. The analysis of these algorithms included highlighting which methods use data reduction (decimation), differences in filtering, any limitations placed upon the algorithms, and in which domain (time versus frequency) these algorithms are calculated.

After the analysis was complete, relationships between each of the algorithms were established that link these similarities and differences together. These methods were then tested using two simulated pseudo-voice signal streams for three different frequencies (HF, VHF, and UHF) in Matlab. Under each frequency the amount of Doppler shift to one of the streams, along with the sampling and decimation rates were varied. The time-delay and Doppler values measured off the ambiguity surfaces for each of the methods were compared to truth to determine the accuracy results, and the number of real computations (additions and multiplies) were calculated for each method for each test case and then contrasted to determine computational complexity. A lengthy error analysis was conducted that points out sources of errors, limitations from the simulation software, as well as possible future points of study regarding these methods.

This thesis provides an analysis that (*i*) discusses similarities among five existing cross-ambiguity function algorithms and (*ii*) test these methods in an emitter location scenario. By testing these methods under a specific emitter location scenario insights were obtained that can be used to (*i*) understand the trade-offs in computational complexity versus accuracy and to (*ii*) determine circumstances under which these methods may be preferred for emitter location. A secondary objective to condense into one document information on these five algorithms was also achieved.

This thesis consists of five chapters. Chapter 2 discusses background information on emitter location and cross-ambiguity functions. Chapter 3 provides the mathematical derivations for each of the five methods remaining as well as the analysis that highlights similarities and differences between each method. Chapter 4 gives the relationships between each method and explains the test setup used and derivation of test case parameters, as well as providing the results and error analysis from the tests conducted. Chapter 5 is the conclusion that includes a summary of the positives and negatives of the five CAF algorithms examined. The References section follows Chapter 5. The Appendices appear after the References section and include the Matlab simulation code used for the test cases performed. Appendix A contains the main driver Matlab script code. Appendices B through G contain code for the Matlab functions used for calculating the TDOA and FDOA and number of computations for each method or for helper functions used in calculation of the latter.

Chapter 2 Background on Ambiguity Function

In this chapter, the background on the narrow-band cross ambiguity function (CAF) is presented from an emitter location perspective. The background information presented includes a definition of the emitter location problem and why the ambiguity function must be used. Within this presentation an alternative view on how the CAF may be thought of as the inner product of two vectors is given. Terms used in estimation theory and signal processing such as range and low-pass equivalent signal are briefly discussed.

2.1 Definition and Theoretical Viewpoint

Emitter location can best be defined by examining Figure 1:

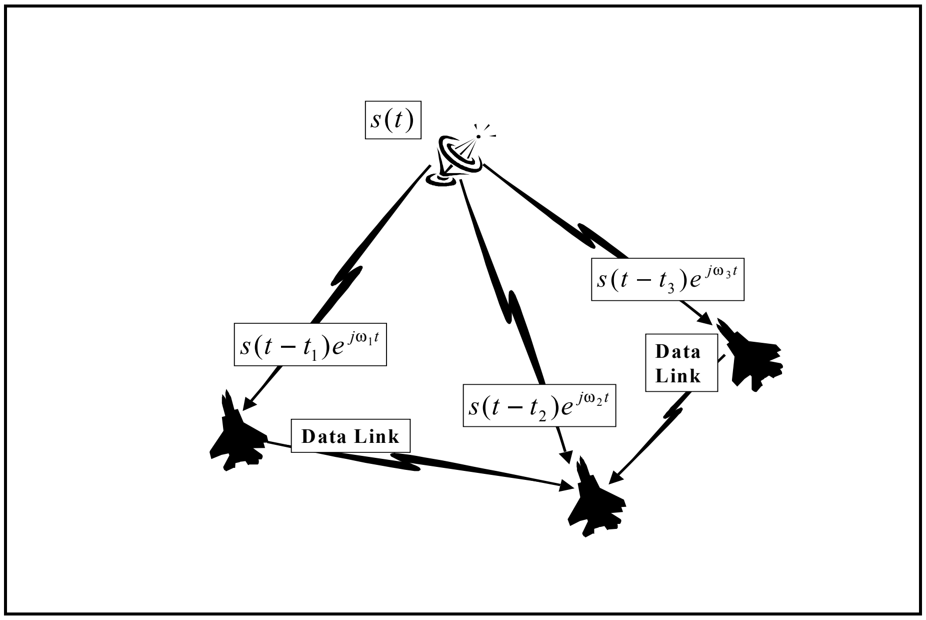
$s(t)$
$s(t-t_3)e^{j\omega_3 t}$
$s(t-t_1)e^{j\omega_1 t}$
Data Link
$s(t-t_2)e^{j\omega_2 t}$
Data Link

Figure 1 Multiple Platform Emitter Location

An emitter transmits a signal denoted by $s(t)$. The data collectors represented by airplanes, collect signal data. Each collector is at a specific distance or range away from the emitter. Since the exact location of the emitter is not known, the distance from each collector to the emitter is not known. The emitter transmits its signal at some time t. Since there is a distance between each collector and the emitter, in some instances thousands of meters, the emitter's signal $s(t)$ arrives at the collectors at some time after time t. This received signal is a delayed version of the transmitted signal. Assuming the emitter is stationary, if the collectors are moving, they each have a velocity with respect to the emitter, so the received signals also have Doppler shifts applied to them. In the case of Figure 1, the three received signals from left to right, including the time delays and Doppler shifts are $s(t\text{-}t_1)e^{j\tau_1 t}$, $s(t\text{-}t_2)e^{j\tau_2 t}$, and $s(t\text{-}t_3)e^{j\tau_3 t}$. Where t_1, t_2, and t_3 represent the time delays and ν_1, ν_2, and ν_3 represent the frequency offsets or Doppler shift of the received signals with respect to the transmitted frequency ω at collectors one, two, and three respectively.

In order to locate the emitter, TDOA and FDOA are applied to the signal data.

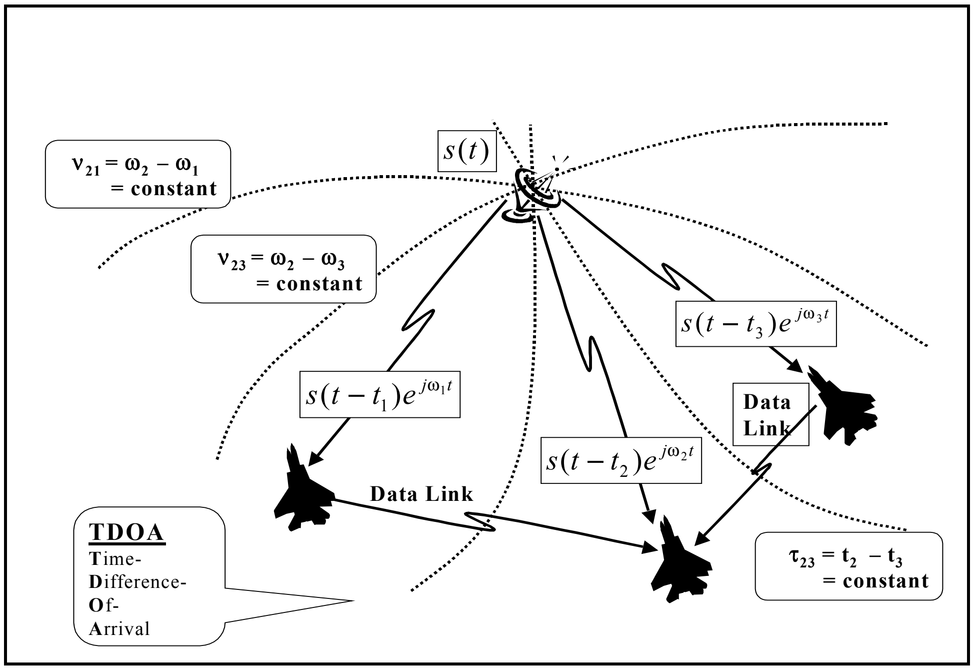
$\nu_{21} = \omega_2 - \omega_1$ = constant
$s(t)$
$\nu_{23} = \omega_2 - \omega_3$ = constant
$s(t-t_3)e^{j\omega_3 t}$
$s(t-t_1)e^{j\omega_1 t}$
Data Link
$s(t-t_2)e^{j\omega_2 t}$
Data Link
TDOA Time-Difference-Of-Arrival
$\tau_{23} = t_2 - t_3$ = constant

Figure 2 TDOA and FDOA Contours

The time of arrivals of the signal data between two collectors are subtracted to produce a difference τ, thus the acronym TDOA. In Figure 2, the TDOA between collectors two (middle plane in the figure) and one (left-most plane) is denoted by τ_{21}, and τ_{23} denotes the TDOA between collectors two and three (right-most plane). The math tells us that a TDOA will produce a hyperbola in which the emitter lies on. This is fine, except where on the hyperbola does the emitter lie? If a second TDOA was computed, another hyperbola could be computed. Where these two hyperbolas intersect would be the emitter location, thus the reason for having τ_{21} and τ_{23}!

Taking the differences between the Doppler shifts between two collectors will produce a FDOA, ν. In Figure 2, the FDOA between collector two and collector one (left-most plane) is denoted by ν_{21}, and ν_{23} denotes the FDOA between collector two and collector three. Like the TDOA technique, FDOA will also produce a hyperbola. So by having three collectors, the emitter could be located using the TDOA technique only or the FDOA technique only, but what if both techniques were used together. In the case of Figure 2 there would be four curves intersecting, thus providing a more accurate emitter location!

By knowing the TDOA and FDOA of the collectors and applying estimation theory techniques an estimate of the emitter's location can be obtained. This is not as easy as is described. The emitter position is what is desired, but some unknowns make this emitter location problem a difficult one to solve. The emitter is transmitting its signal. It is received at some time offset from when it was transmitted, but at what time did the emitter begin transmitting? This is one unknown! The other unknown is the received signal is received at some frequency, which due to collector velocity has a Doppler shift on it, but what frequency is the emitter transmitting at? If the time at which the emitter began transmitting was known, the time delay at each platform could be computed accurately and the TDOAs could easily be computed, and the same goes for FDOAs if the frequency of the emitter was known, but since the time at which the emitter began transmitting and the frequency at which it is transmitting is unknown, how is this emitter going to be located if TDOA and FDOA cannot be computed?

The answer is simple, correlation. By computing the CAF between collectors one and two and then two and three, τ_{21}, τ_{23}, ν_{21}, and ν_{23} can be computed. The CAF can be computed by taking the cross-correlation between the received signals at two platforms. The cross-correlation takes one of the received signals at one of the collectors and it delays it a certain amount. That delayed signal is then slid a certain amount in time with respect to another received signal, from a different collector, which is kept fixed in time. At each of the slide amounts, the samples from the fixed and shifted signal are multiplied and summed together, producing a number. The maximum number over the entire duration of the shifted and the fixed signal is the most the two signals match. If a cross-correlation is performed in two-dimensions, with the two dimensions being TDOA and FDOA, a peak will be produced. The received signals are the same signal only with a different time delay, amplitude, and Doppler shift. The point where the maximum value from the cross-correlation occurs (the peak) will have a corresponding TDOA and FDOA with it. It is this TDOA and FDOA that is of interest. This TDOA and FDOA are the ones that we are trying to calculate!

To be able to obtain the received signals in a form to be able to cross correlate them, some manipulation must be performed. The first manipulation is to generate the complex envelope for the received signals. To do this, first examine what types of signals the collectors intercepted. The signals that were collected by the collectors were only collected for the finite duration of time *T*. This makes the signals time-limited. Recalling properties in signal processing theory, time-limited signals produce signal replicas of the signal in the frequency domain. The data collected by the collectors was real data. This will produce spectra pairings at both positive and negative frequencies in the frequency domain

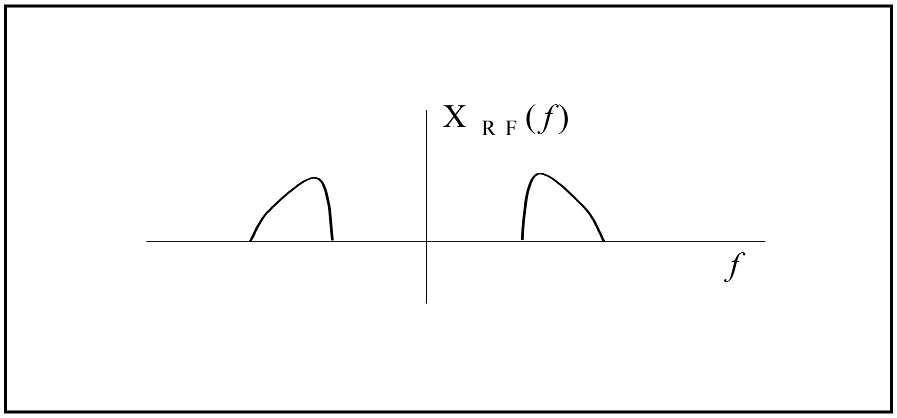

Figure 3 Frequency Domain View of Received Signal

To generate the complex envelope, modulate the positive or negative frequency band to zero by multiplying by a complex exponential, $e^{j\omega}$ in the time domain. This multiplication shifts the spectra in the frequency domain. The sign of the $j\omega$ term of the modulating complex exponential is determined on whether the positive (sign is negative for a shift of the spectra down in frequency) band or negative (sign is positive for a shift of the spectra up in frequency).

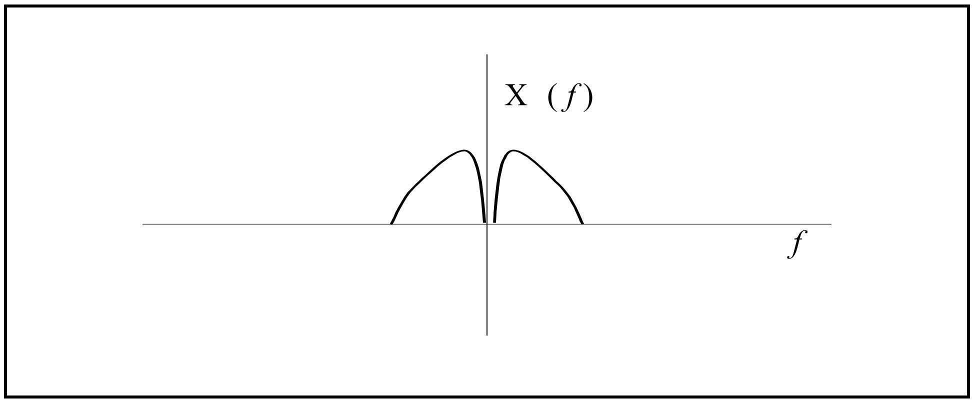

Figure 4 Frequency Domain View of Received Signal Modulated to DC

For purposes of this paper, the positive band is shifted towards zero. Once the positive band is shifted to DC or zero, a low pass filter filters out all of the other spectral bands/replicas. This now has produced a signal that has a bandwidth from $-B/2$ to $B/2$ and is centered at frequency equal to zero.

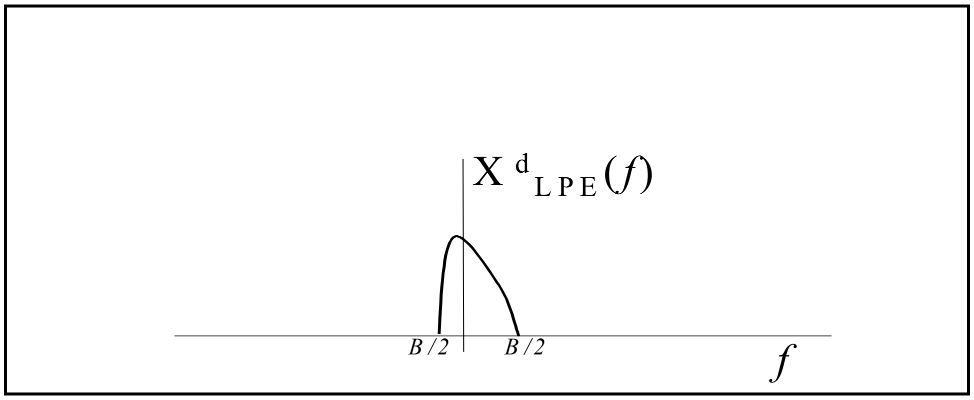

Figure 5 Low Pass Equivalent (Complex Envelope) Signal

This is called the low-pass equivalent (LPE) or complex envelope. Generating the complex envelope has advantages. Using complex models provides insight that working with real signal models would not allow. It is this insight that will allow us to exploit the ideas needed to be able to produce the CAF. Another advantage of using the complex envelope is that because the signal takes up the frequency band from $-B/2$ to $B/2$, the signal can be sampled at $F_s \geq B$ Hz. If the real signal model was used the sampling rate would have to be $2B$. Having to sample at a lower rate requires less time and less calculations.

The second manipulation comes in the form of applying the time delay and Doppler imparted to the collected signal.

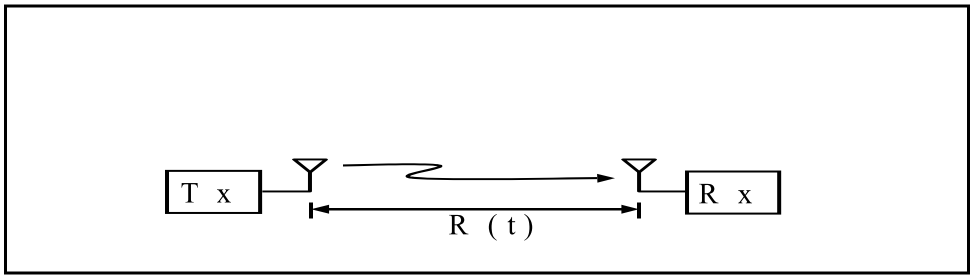

Figure 6 Model of Transmitter (Emitter) and Receiver (Collector)

In Figure 6 the emitter from Figure 1 is the transmitter (Tx) and a collector from Figure 1 is represented as a receiver (Rx). The general time delay, also called propagation time, denoted by $\tau(t)$, is a function of time. This value calculated by taking the distance from the emitter to the collector and dividing it by the speed of light is given by

$$\tau(t) = \frac{R(t)}{c}, \tag{1}$$

where c is the speed of light constant. The general range is denoted by $R(t)$. Taking the Taylor series expansion of the distance expands it into the following form

$$R(t) = R_o + vt + \frac{at^2}{2} + \ldots, \tag{2}$$

Emitter location is done for some specified observation interval. This interval is usually small in time for a variety of reasons; mainly the emitter does not usually transmit for a continuous period of time but for short durations. Therefore, since the observation interval is small, it is assumed that the velocity is relatively constant over this observation interval. From this assumption, the third, and higher terms of $R(t)$ approximate zero and can be ignored. This leaves $R(t)$ as a function of R_o and vt

$$R(t) = R_o + vt\,. \tag{3}$$

Applying the time delay and Doppler to the received signals yields a general form as

$$s_r(t) = s(t - \frac{[R_o + vt]}{c}) = s([1 - \frac{v}{c}]t - \frac{R_o}{c})\,. \tag{4}$$

This signal is the received band pass signal; it has not yet been shifted to create the LPE (complex envelope) signal. What is interesting about this signal is the $[1 - v/c]t$ term alters the received signal. Based on the value of Doppler, v, the signal is time scaled. In the time domain, the received signal is compressed if v is negative, or the signal is expanded if v is positive.

Also, by examining the above equation for $s_r(t)$, the time delay has been simplified to a constant (no longer is it dependent on time t as was previously shown). The range, R_o can be calculated by

$$R_o = \sqrt{(X - X_E)^2 + (Y - Y_E)^2 + (Z - Z_E)^2}\,, \tag{5}$$

where X, Y, Z are the coordinates of a collector at a specific time of interest, and X_E, Y_E, Z_E are the coordinates of the emitter. The problem with the above equation can easily be seen. The emitter location is not known! Simplification cannot be done. However, letting $\tau_d = R_o/c$ makes the equation for $s_r(t)$ become

$$s_r(t) = s([1 - \frac{v}{c}]t - \tau_d)\,, \tag{6}$$

where (6) is a simpler form to view than (4).

To understand how the time delay, Doppler, and complex envelope all come together, the analytic signal must be analyzed

$$\tilde{s}(t) = E(t)e^{j[\omega_c t + \varphi(t)]}\,, \tag{7}$$

where $E(t)$ equals $s(t)$ and $e^{j[\omega_c t + \phi(t)]}$ is the modulation required to produce a LPE (complex envelope) signal. Applying the Doppler shift and time delay to the analytic signal yields:

$$\tilde{s}_r(t) = \tilde{s}([1-\frac{v}{c}]t - \tau_d) = E([1-\frac{v}{c}]t - \tau_d)e^{j\{\omega_c([1-\frac{v}{c}]t - \tau_d) + \varphi([1-\frac{v}{c}]t - \tau_d)\}}.$$

(8)

Analyzing the $[1 - v/c]$, it can be seen that the term $[1 - v/c] \approx 1$. This is because the speed of light constant, c approximates 3x10^8 m/s. Any velocity substituted for v is always << 3x10^8 m/s. Performing the division of v/c will produce a result that ≈ 0. Now if it is assumed that $E(t)$ and $\phi(t)$ vary slowly enough for the range of Doppler of interest such that they remain constant over the observation interval, then $E(t)$ can be approximated as

$$E([1-\frac{v}{c}]t) \approx E(t), \tag{9}$$

and $\phi(t)$ can be approximated as

$$\varphi([1-\frac{v}{c}]t) \approx \varphi(t). \tag{10}$$

This is called the narrowband approximation. Note that this approximation can be used if the signal of interest that the emitter is transmitting is indeed a narrowband signal. For the purposes of this paper, it is assumed that the transmitted signal is narrowband. If the signal is a wideband signal, this approximation cannot be performed!

Using the narrowband approximation, the analytic signal model becomes

$$\tilde{s}(t) = E(t-\tau_d)e^{j\{\omega_c t - \omega_c(\frac{v}{c})t - \omega_c\tau_d + \varphi(t-\tau_d)\}} = E(t-\tau_d)e^{j\varphi(t-\tau_d)}e^{-j\omega_c\tau_d}e^{-j\omega_c\frac{vt}{c}}e^{j\omega_c t},$$

(11)

where $E(t_-\tau_d)e^{j\phi(t-\tau_d)}$ is the received signal's LPE (complex envelope) signal time-shifted by τ_d, $e^{j\omega_c t}$ is the carrier term, $e^{j\omega_c\tau_d}$ is a constant phase term and $e^{-j\omega_c vt/c}$ is the Doppler shift term. If the analytic signal is simplified by letting $\alpha = \omega_c\tau_d$, $\omega_d = v/c$, and $\tilde{s}(t - \tau_d) = E(t - \tau_d)e^{j\varphi(t-\tau_d)}$ then the narrowband LPE signal model becomes

$$\tilde{s}_r(t) = \hat{s}(t-\tau_d)e^{j\alpha}e^{-j\omega_d t}\,. \tag{12}$$

To estimate Doppler and time delay, first consider a continuous time view, for simplicity. This paper assumes that the signals mentioned in the next chapters to try and explain the procedure for estimating Doppler and time delay are all LPE signals, but $\hat{s}$ is not used. Given LPE signals $s_1(t) = s(t)$ and $s_2(t) = s(t-\tau_d)e^{j\alpha}e^{-j\omega_d t}$ for $t \in [0,T]$, compute τ_d (time delay) and ω_d (Doppler). One way to view this is to visualize vectors in two-dimensional vector space. In specific consider two vectors, V_1 and V_2 and let them be separated by some distance, call id θ_d.

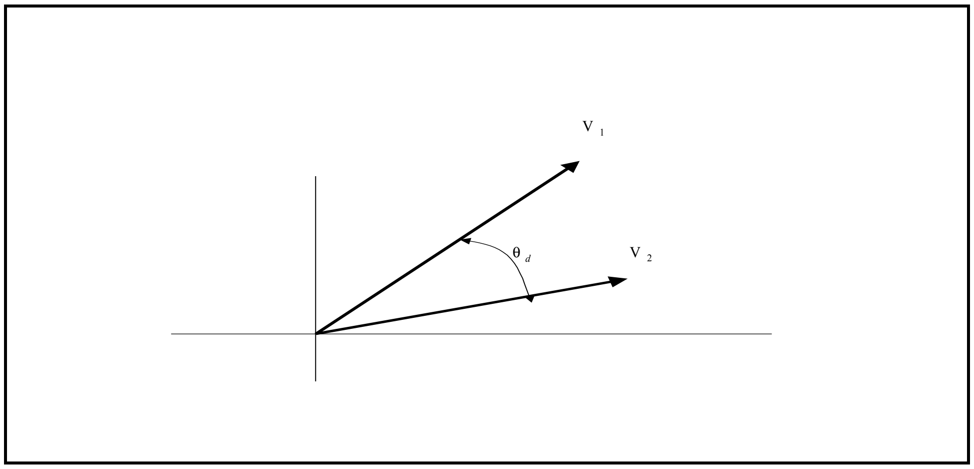

Figure 7 Two-Dimensional Vector View of Cross Correlation

The two vectors are shown above in Figure 7. Let the goal be to measure θ_d. One way of finding θ_d is to keep V_1 fixed and rotate V_2 clockwise by some small increment θ. For each value of θ, compute an inner product between the two vectors by

$$\langle V_1, V_2(\vartheta)\rangle = A(\vartheta)\,. \tag{13}$$

This will produce a result that will be a function of θ. Notice what value produces the largest inner product. The maximum value of $A(\theta)$ occurs when V_1 and V_2 are directly on top of one another or when θ equals θ_d. So if $A(\theta)$ is plotted over all values of θ, the maximum would be at the peak of $A(\theta)$! This sounds very familiar to what the CAF does. In fact the inner product just described is mathematically equivalent to the CAF!

To represent the CAF mathematically, first, let $s_{\omega,\tau}(t) = e^{j\omega t}s_2(t+\tau)$. All that has been done is taken $s_2(t)$ and delayed it in time by τ and shifted it in frequency by some amount ω. So the new form of $s_{\omega,\tau}(t)$ becomes

$$s_{\omega,\tau}(t) = s(t-\tau_d+\tau)e^{j\alpha}e^{j(\omega-\omega_d)t}\,. \tag{14}$$

Performing the inner product obtains

$$A(\omega,\tau) = \langle s_1(t)s_{\omega,\tau}(t)\rangle\,. \tag{15}$$

Using the definition of inner product, $A(\omega,\tau)$ can be rewritten as

$$A(\omega,\tau) = \int_0^T s_1(t)\overline{s_{\omega,\tau}(t)}dt\,. \tag{16}$$

Upon substituting the value for $s_{\omega,\tau}(t)$ and taking the complex conjugate, the following equation for $A(\omega,\tau)$ is given by

$$A(\omega,\tau) = \int_0^T s_1(t)\overline{s(t-\tau_d+\tau)}e^{j(\omega-\omega_d)}e^{j\alpha}dt\,, \tag{17}$$

where $\alpha = -\omega_c\tau_d$. From the inner product view, $|A(\omega,\tau)|$ has a maximum at $\omega = \omega_d$ and $\tau = \tau_d$. Calculating the maximum value produces

$$|A(\omega,\tau)| = \left|\int_0^T s(t)\overline{s(t-\tau_d+\tau_d)}e^{j\alpha}e^{-j(\omega_d-\omega_d)t}dt\right| = \int_0^T |s(t)|^2 e^{j\alpha}dt = BE_s = Energy\,,$$

(18)

where noting that $e^{j\alpha}$ is a constant and is represented by B (not bandwidth). By finding the peak from the CAF, the energy of the signal can be calculated! The process for computing the CAF is becoming defined.

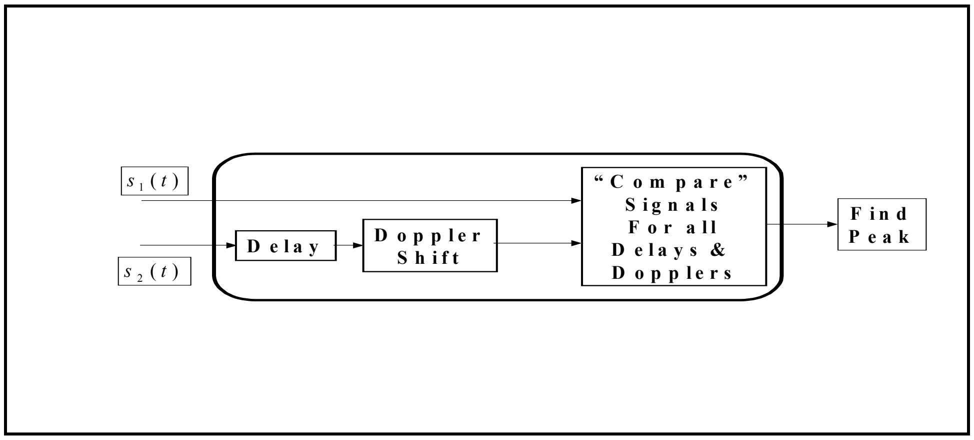

Figure 8 Coarse Process for Calculating CAF

To compute a CAF, take two signals. Each signal will be from a different collector. On the second signal, impart a delay and a Doppler Shift. Then take the inner product of the two signals for all the time delays and Dopplers of interest. What comes out is the CAF

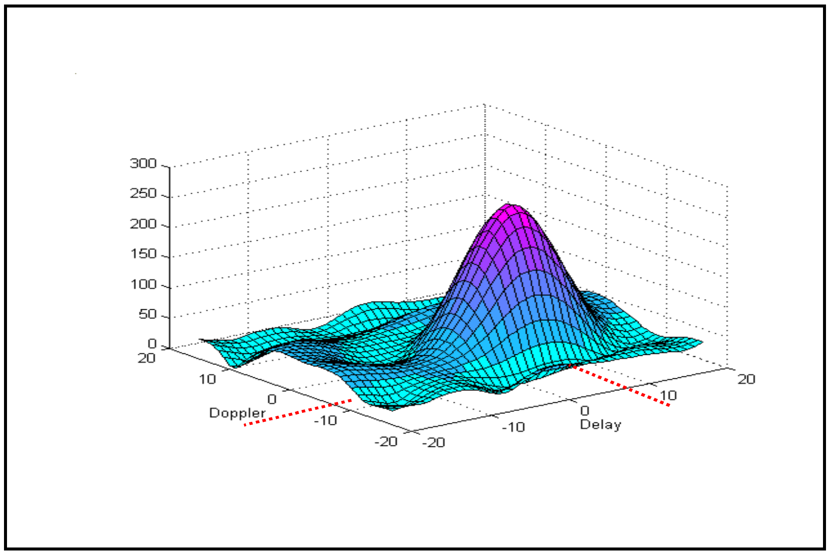

Figure 9 Example of Output CAF Surface

A peak will result, and this peak will be centered on the true time delay and the true Doppler.

What is interesting about the CAF is that not only will it provide the time delay and Doppler measurements, but it also provides insight into the characteristics of the time delay and Doppler. Start with examining the time delay. First begin by considering the case when $\omega = \omega_d$ (the CAF has been centered on the Doppler value, but not time delay).

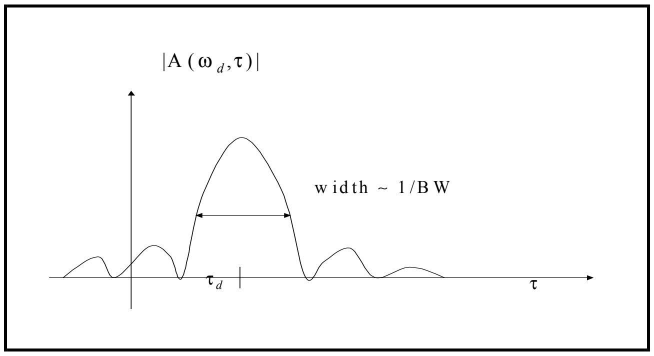

Figure 10 Cross-Section of CAF in Time delay Direction

This CAF is represented mathematically by

$$\left|A(\omega,\tau)\right| = \left|\int_0^T s(t)\overline{s(t-\tau_d+\tau)}e^{j\alpha}dt\right|. \tag{19}$$

By examining the right-hand side of (19), the TDOA is nothing more than a correlation between two signals. The most interesting insight that can be taken from Figure 10 is that the slice of the CAF in the time delay direction shows that TDOA or time delay peak width is based on inverse of the bandwidth of the signal the emitter transmits. The CAF provides information on the signal that was transmitted. If the bandwidth of the signal is small, the CAF peak in the TDOA direction will have a wide lobe, where if the bandwidth of the transmitted signal was wide, then the peak in the TDOA direction will be a sharp, skinny peak.

Similar information can be obtained from the Doppler part of the CAF. To look at the Doppler, let $\tau = \tau_d$ (TDOA portion of CAF peak has been found, but not Doppler). This CAF is as follows:

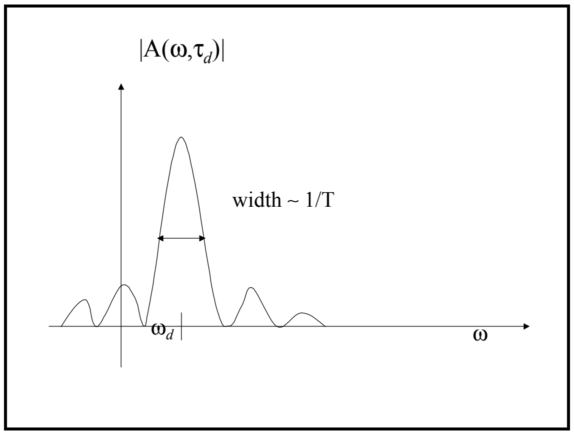

Figure 11 Cross-Section of CAF in Doppler Direction

This CAF is mathematically represented by

$$\left|A(\omega,\tau)\right| = \left|\int_0^T \left[\left|s(t)\right|^2 e^{j\alpha} e^{j\omega_d t}\right] e^{-j\omega t}\,dt\right|, \tag{20}$$

The right hand side of the (20) depicts that the Doppler or FDOA of the CAF is nothing more than the Fourier Transform of the complex sinusoid $e^{j\omega dt}$, with $|s(t)|^2 e^{j\alpha}$ being a window function! The most important aspect gleamed from the FDOA slice of the CAF is that the width of the FDOA peak is inversely proportional to the time width or integration period of the signal. The longer the integration period (i.e. the more data applied to the CAF) the sharper and narrower the peak will be. A very short integration time will yield a wide lobe in the FDOA direction of the CAF. There is a tradeoff here. The signal of interest collected is usually on for a short period of time and therefore is collected for that short period or for even a shorter period of time. So to get the best FDOA resolution, most of the signal has to be applied to the CAF, so the integration time is as long as possible. This has the possibility to create many calculations.

The equations given for $|A(\omega, \tau)|$, have been for continuous time. Since for continuous time, there is an infinite number of points to process, the CAF, can only be computed for discrete values of time delay and Doppler. To compute the discrete values of time delay and Doppler, some a priori information is needed which includes the max/min Doppler (from largest expected velocity difference) and max/min time delay (from largest expected range difference). The max/min values allow for computation of a precise number of Dopplers and time delays that are of interest to use in computation of the CAF. Two other important values needed are the delay spacing (taken from expected/measured signal bandwidth) and the Doppler spacing (taken from the observation time interval, *T*). These values dictate how fine a grid of time delays and Dopplers that will be used in conjunction with the number of time delays and Dopplers. Once the number of delays, Dopplers, and bin spacing for both are calculated, an easy approach to calculating the CAF is to view it as the Fourier Transform of lag products. Recall that

$$A(\omega,\tau)=\int_0^T s_1(t)\overline{s_2(t+\tau)}e^{j\omega t}e^{j\alpha}dt\,. \tag{21}$$

This is nothing more than the Fourier transform of the products $s_1(t)$ and $s_2(t)$, except $s_2(t)$ has a time delay associated with it, thus the term lag products. If the lag products are defined as

$$f_\tau(t)=s_1(t)\overline{s_2(t+\tau)}\,. \tag{22}$$

Then for each time delay, τ_m of interest, $A(\omega,\tau_m)=F\underline{\{f_{\tau m}(t)\}}$. Where $F\{f_{\tau m}(t)\}$ is the Fourier transform of the lag products. For the corresponding discrete time signals, $f_\tau(t)$ becomes

$$f_m[n]=s_1[n]\overline{s_2[n+m]}\,, \tag{23}$$

where *n* is the sample index and *m* is the delay index, which is synonymous with τ_m in the continuous time world. The delay spacing is set by the sampling interval. The sampling rate is chosen so as to match the signal bandwidth according to the Nyquist rate. This involves having the signal data sent through an A/D converter and then performing preprocessing of the output. This is done for both streams that are to be correlated.

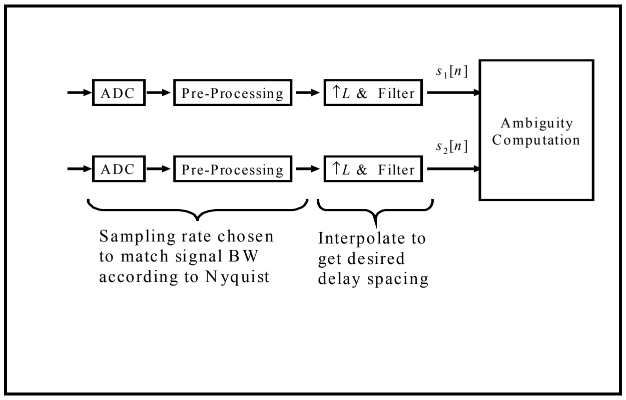
ADC
Pre-Processing
↑L & Filter
$s_1[n]$
ADC
Pre-Processing
↑L & Filter
$s_2[n]$
Ambiguity Computation
Sampling rate chosen to match signal BW according to Nyquist
Interpolate to get desired delay spacing

Figure 12 Process for Computing CAF in Discrete Time

The pre-processed signal data is then interpolated by a factor of *L* and then filtered to produce the proper delay spacing. The two discrete time signal are then used in the computation of the CAF

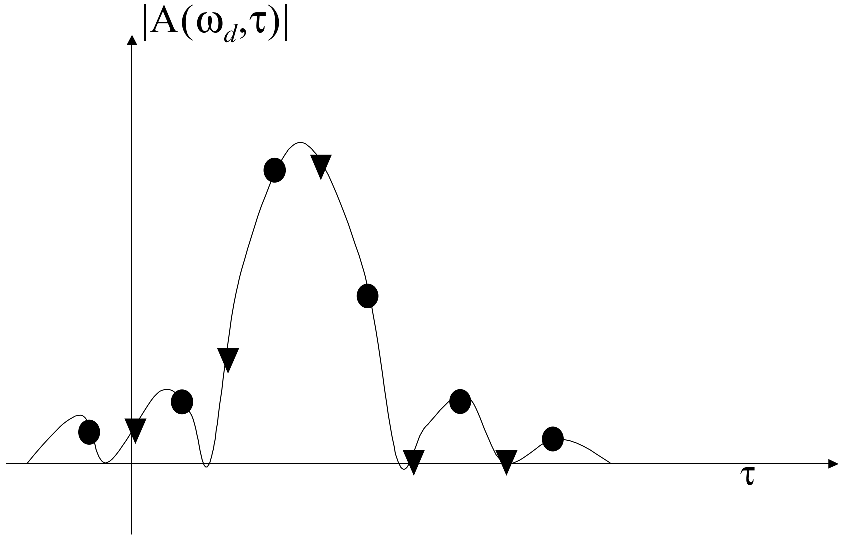

Figure 13 CAF Computed in Discrete Time

As seen in Figure 13, the dots and triangles represent the time delay and Doppler values that the CAF was computed over in discrete time. The solid line connecting the dots and triangles represent what the CAF looks like in continuous time.

Chapter 3 Computationally Efficient Methods

This chapter will develop several computationally efficient methods for computing the CAF. The developments are structured in such a way to clearly show relationships between methods; readers whose interest lies mostly in seeing the final resulting methods rather than the interconnected developments can skip immediately to Chapter 4.3.

3.1 Fourier Transform of Lag Product Viewpoint

The Cross-Ambiguity function can be viewed as a Fourier Transform of lag products. To see this viewpoint, start with the DTFT version of the ambiguity function. In practice the DFT form would be used, but here the DTFT form simplifies notation. The ambiguity function to be computed is given by

$$A(\tau,\nu)=\sum_{n=0}^{N-1} s_1[n]s_2^*[n+\tau]e^{\frac{-j2\pi\nu n}{F_s}}, \tag{24}$$

where τ is an integer and represents the time delay over the range from $0 \le \tau \le T$ and ν is a real variable that represents Doppler. The * denotes complex conjugate. Though the equation in (24) is mathematically correct, the ambiguity function that is computed using this equation is a "brute force" method where no data reduction is performed. Though this method is a "brute force" method, it may be advantageous in some applications.

The $s_1[n]s_2^*[n+\tau]$ term from the summation in (24) is called the lag product, where the lag is the time delay of amount τ between s_1 and s_2. The product arises from the fact that s_1 and s_2 are multiplied together. By letting the lag product be equal to

$$r_\tau[n]=s_1[n]s_2^*[n+\tau], \tag{25}$$

and substituting it back into (24), it can be seen that the ambiguity function is nothing more than the DTFT of the lag product

$$A(\tau,\nu)=\sum_{n=0}^{N-1} r_\tau[n]e^{\frac{-j2\pi\nu n}{F_s}} \quad . \tag{26}$$

When computing the DTFT in (26) directly, the DTFT will be over the range from -F_s/2 to F_s/2 (standard range for a DTFT). However, in virtually ever case in practice, the maximum Doppler shift is << than F_s/2. So when the DTFT is implemented using the DFT, the majority of the DFT points that are computed are not even in the desired range of Doppler. Therefore, a method is desired to find a more efficient way to compute (26), which will be shown below. The first step is to simply re-index the sum in (26) using $n = mL+p$, where $m = 0 \ldots M$-1 and $p = 0 \ldots L$-1. To ensure that M blocks are created, $r[n]$ must be made to be a length divisible by L; this is done by zero-padding $r[n]$ to the desired length. The value of L is a parameter of the algorithm that can be chosen. The result in (26) now becomes

$$A(\tau,\nu)=\sum_{m=0}^{M-1}\sum_{p=0}^{L-1} r_\tau[mL+p]e^{\frac{-j2\pi\nu(mL+p)}{F_s}} \quad . \tag{27}$$

Separating the complex exponential in (27) and rearranging terms gives

$$A(\tau,\nu)=\sum_{m=0}^{M-1} e^{\frac{-j2\pi\nu mL}{F_s}} \sum_{p=0}^{L-1} r_\tau[mL+p]e^{\frac{-j2\pi\nu p}{F_s}} \quad . \tag{28}$$

To simplify this further, note the following. The Doppler model states that

$$f = f_o\frac{v}{c}, \tag{29}$$

where f is the Doppler shift in Hz, f_o is the frequency of the signal of interest in Hz, v is the velocity of the collection platform, and c is the speed of light constant. Assuming an extreme situation, that v is at most

1000 m/s (supersonic aircraft) and f_o is at most 10 GHz, then the dynamic range of the Doppler shift becomes

$$f = 10^{10}\frac{10^3}{10^8} = 0.1\,\text{MHz}\,, \tag{30}$$

which is much smaller than the bandwidth of the typical signal centered at 10 GHz. Since the collected signals are complex envelope, the sampling rate (F_s) of either collected signal (s_1 or s_2), according to Nyquist's sampling theorem, is 10 GHz. Comparing the maximum Doppler shift to F_s, the F_s is 10^5 times greater than the Doppler shift, f. Then the complex exponential on the inner summation of (28) is

$$e^{-j2\pi(10^{-5})L} \approx 1\,, \tag{31}$$

for values of L up to quite large values. Therefore, the approximation can be made that

$$e^{\frac{-j2\pi\nu p}{F_s}} \approx 1, p = 0,1,\ldots L-1\,, \tag{32}$$

which says that if L is selected such that $L << F_s/2\pi\nu_{max}$, where ν_{max} is the maximum expected Doppler shift, an upper bound can be placed on the block size L based on the sampling rate and the maximum expected Doppler. This simplifies the ambiguity function in (28) to

$$A(\tau,\nu) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi\nu mL}{F_s}} \sum_{p=0}^{L-1} r_\tau[mL+p]\,. \tag{33}$$

Although not discussed, for a method of determining $A(\tau,\nu)$ without making the approximation in (32) see Section 5.7.2 on page 153 of [9].

Noticing that the lag product of the inner summation in (33) is multiplied by nothing by ones, this gives

$$A(\tau,\nu)=\sum_{m=0}^{M-1} e^{\frac{-j2\pi\nu mL}{F_s}} \sum_{p=0}^{L-1} r_\tau[mL+p]1[p], \tag{34}$$

where 1[p] is defined as:

$$1[n]=\begin{cases}1, & n=0,1,\ldots,L-1\\ 0, & \text{otherwise}\end{cases}, \tag{35}$$

where L is the chosen block length. After a change of variable in the inner summation of (34), (34) becomes

$$A(\tau,\nu)=\sum_{m=0}^{M-1} e^{\frac{-j2\pi\nu mL}{F_s}} \sum_{n=mL}^{mL+L-1} r_\tau[n]1[n-mL]. \tag{36}$$

This is exactly the result proposed in Section V-A of [1] for "fine mode" computation, although the method of development was quite different (it was based on rough arguments related to filtering and decimation). The result in (36) can be viewed as a three step process for each delay value τ: (*i*) form the lag product $r[n]$ given in (25), (*ii*) apply a filter of length L – as in the inner summation of (36), and (*iii*) compute the DFT of the filtered sequence – this DFT would play the role of the DTFT in the outer sum in (36). The length L filter can be viewed as filtering using a rectangular impulse response of length L, given by (35) and then decimating by a factor of L; note that here the filter length and the decimation factor are the same. In practice the inner summation in (36) can be defined as:

$$\tilde{r}_\tau[m]=\sum_{n=mL}^{mL+L-1} r_\tau[n], \tag{37}$$

which is the resulting output of filtering $r[n]$ using (35) and then decimating by L. The decimation factor L is computed as $L = F_s/\nu_{max}$, where, as mentioned previously, F_s is the sampling rate of the received signal and ν_{max} is the maximum expected Doppler shift expected on the received signal of interest. Thus by substituting (37) into (36), (36) can be re-written as

$$A(\tau,\nu)=\sum_{m=0}^{M-1}\widetilde{r}_{\tau}[m]e^{\frac{-j2\pi\nu mL}{F_s}} \quad . \tag{38}$$

Note that because the filter length equals the decimation factor, each output of this filter and decimation is computed by moving the length-L filter ahead by L samples. This results in the non-overlapped blocks of (36).

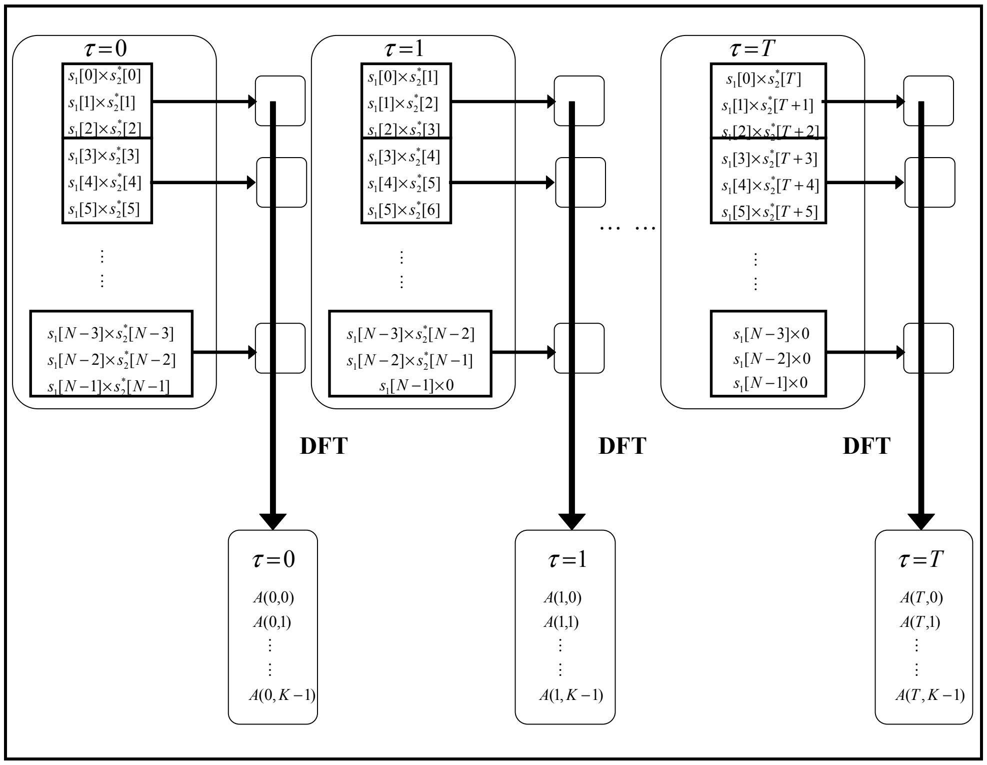

Figure 14. Computation of Ambiguity Function Using "Fine-Mode"
Used with Permission From [10]

Procedures:

1. Determine the number of time delays and Doppler shifts of interest.
2. *Design* an "all-ones" filter $1[n]$ based on the number of Doppler shifts of interest. Length of the filter (L) has an upper limit of $F_s/2\pi v_{max}$, where v_{max} is the max Doppler shift frequency.
3. Set the decimation rate $D = L$.
4. Zero-pad $s_1[n]$ and $s_2[n]$ to a length that is divisible by the length of $1[n]$ (L).
5. Determine the number of blocks M = length($s1[n]$)/L.
6. Compute each Inner Sum and take the DFT

```
for $\tau = 0$ to T
  for $m = 0$ to $M-1$
    /* Get data for this block */
    $\tilde{r}_\tau[m] = \sum_{n=mL}^{mL+L-1} r_\tau[n]1[n-mL]$ , where $r_\tau[n] = s_1[n]s_2^*[n+\tau]$ is the lag product

    Zero-pad $\tilde{r}_\tau[m]$ to get appropriate Doppler bin spacing
  end for on $m$

  $A(\tau,k) = \sum_{m=0}^{K-1} \tilde{r}_\tau[m] e^{\frac{-j2\pi kmL}{K}}$ , for $k = 0, 1, 2, \ldots, K$

  /* $A(\tau,k)$ is nothing more than the DFT of the filtered and decimated lag- product */
end for on $\tau$
```

Figure 15. Procedures for Computing the Ambiguity Function Using "Fine-Mode"

The method in (38) uses an "all-ones" rectangular window. This is not the most optimal filter to use in certain applications, but can often be used successfully despite its poor qualities as a filter. For example, assume that for the correct time delay (τ), the ambiguity surface looks like the following

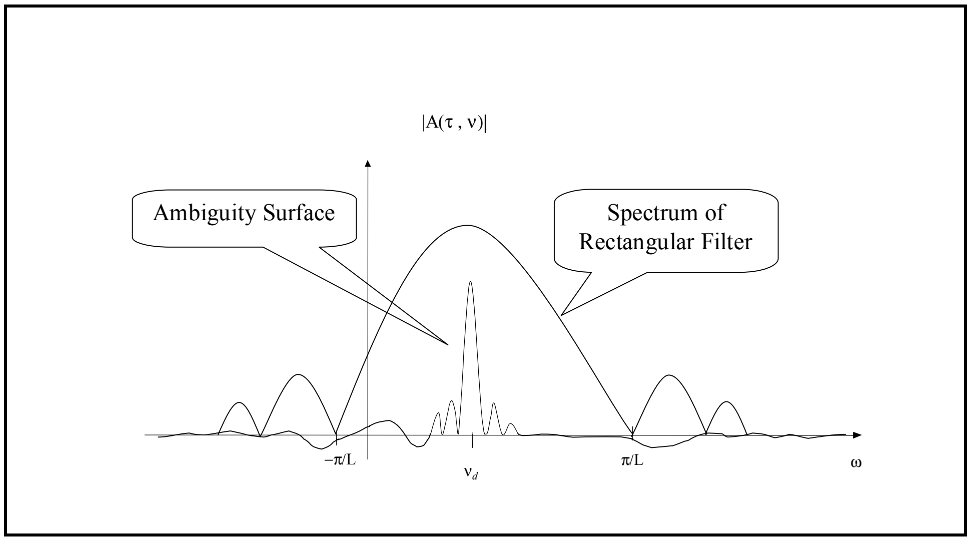

Figure 16 Aliasing from Rectangular Window

Where the ambiguity surface is centered on its true Doppler (ν_d), and the Doppler range of interest is from +/- π/L. The sinc function shown in Figure 16 is the DTFT of the rectangular filter that filters the lag product. It does not go to zero outside of the Doppler range of interest. Such, there is aliasing as parts outside the Doppler range of interest get folded back in after decimation. To see this with a more specific example, assume that the length of the rectangular filter is L = 100, the decimation rate M = 100, the Doppler range of interest is -5 kHz $\leq \nu \leq$ 5 kHz, and the sampling rate of the signal Fs = 1 MHz. The frequency response of this rectangular filter is that of Figure 17. Zooming in to view the passband of the rectangular filter, it is clear from Figure 18 that the passband covers the Doppler range of interest. Afte.r decimating by M, frequency shifted replicas of the filter appear at multiples of F_s/M = 10 kHz. These replicas are aliased back into the passband of the rectangular filter centered at f = 0 Hz, as is seen in Figure 19. Even though the rectangular filter is not an ideal low-pass filter, it has a nice property that all the nulls from the aliasing are in the center of the passband or equivalently at the center of each Doppler bin [10], which minimizes the impact of the aliasing.

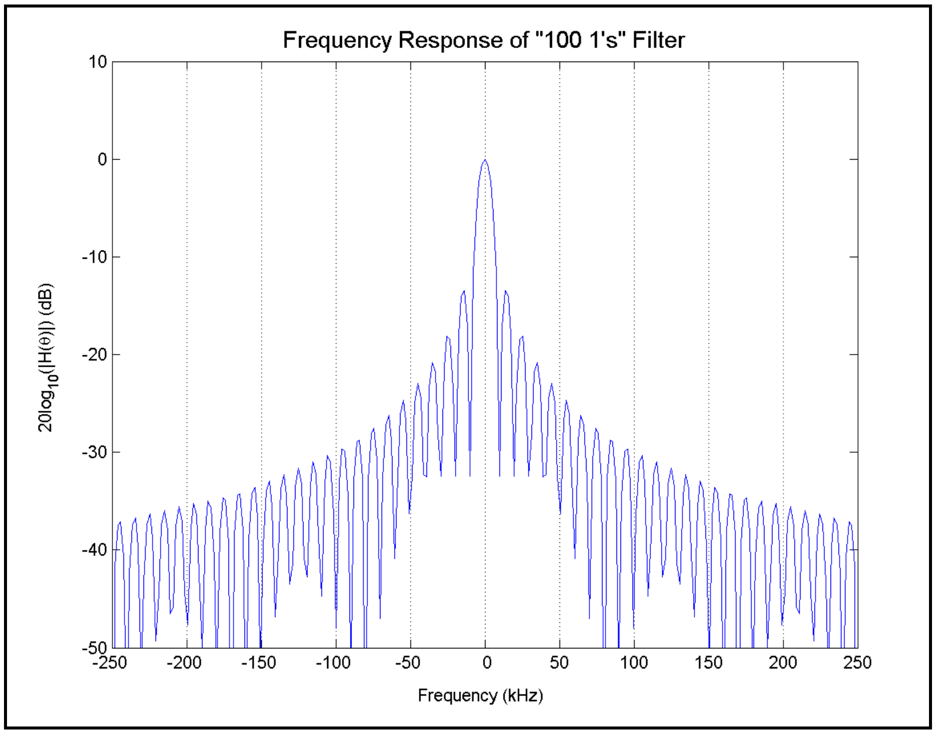

Figure 17 Frequency Response of a Rectangular Filter of Length *L* = 100. Used with Permission from [10]

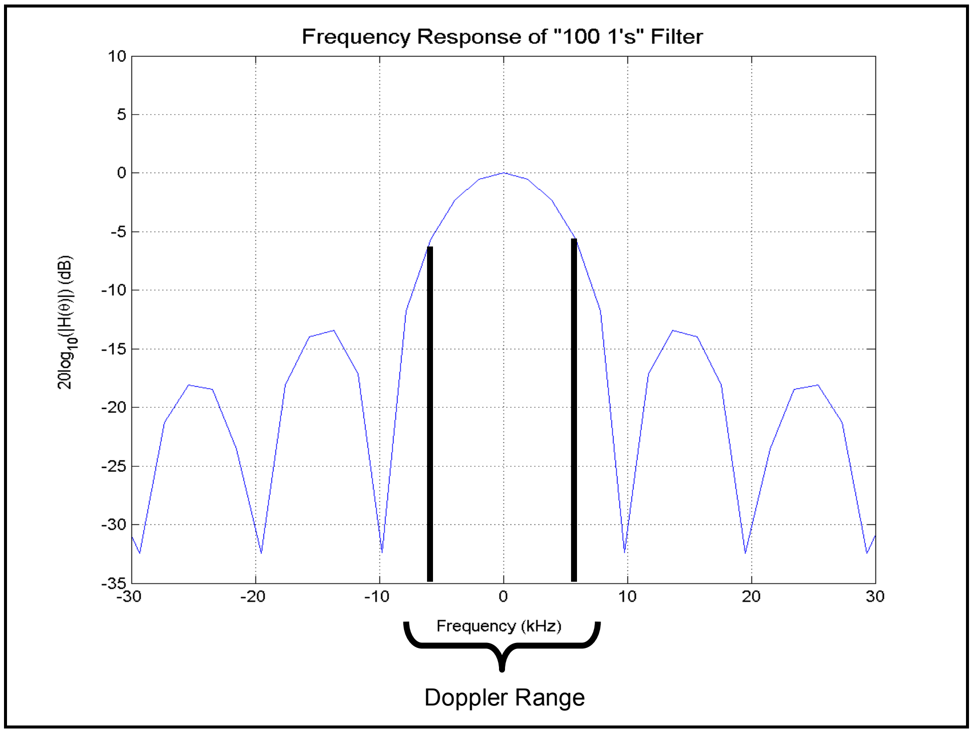

Figure 18 Passband of Rectangular Filter Covering the Doppler Range

Used with Permission from [10]

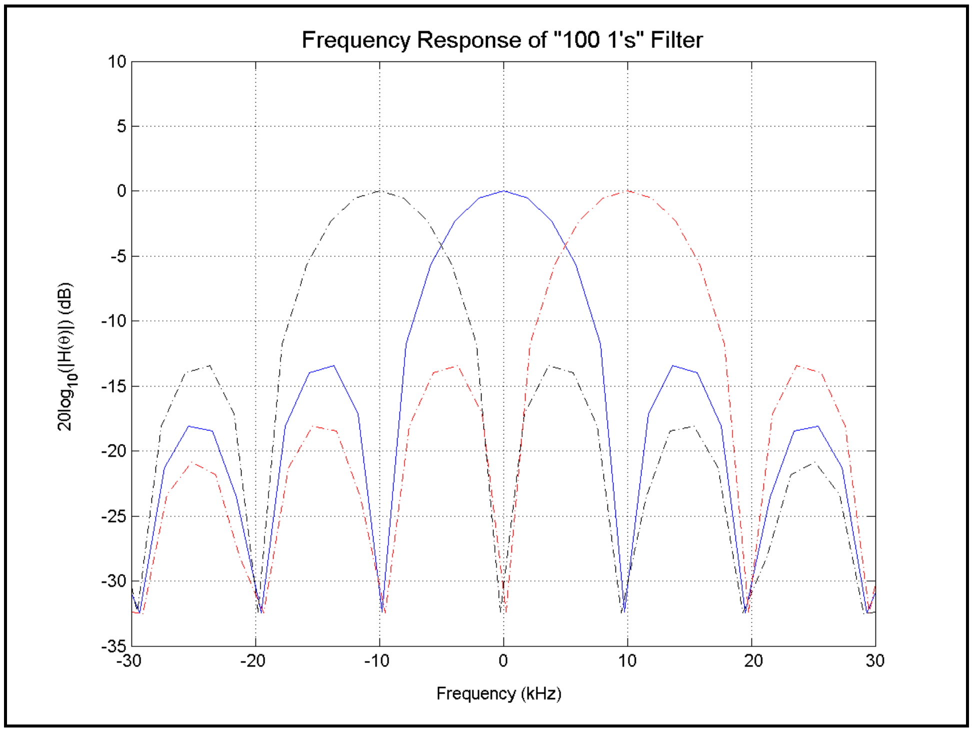

Figure 19 Aliasing of Rectangular Filter

Used with Permission from [10]

In most cases, a more generic filter that is specifically designed for the Doppler range of interest is desired. This design should be carried out in the frequency domain to produce an optimal generic filter that has a flat pass-band and a more narrow transition band, thus to avoid the effects of aliasing.

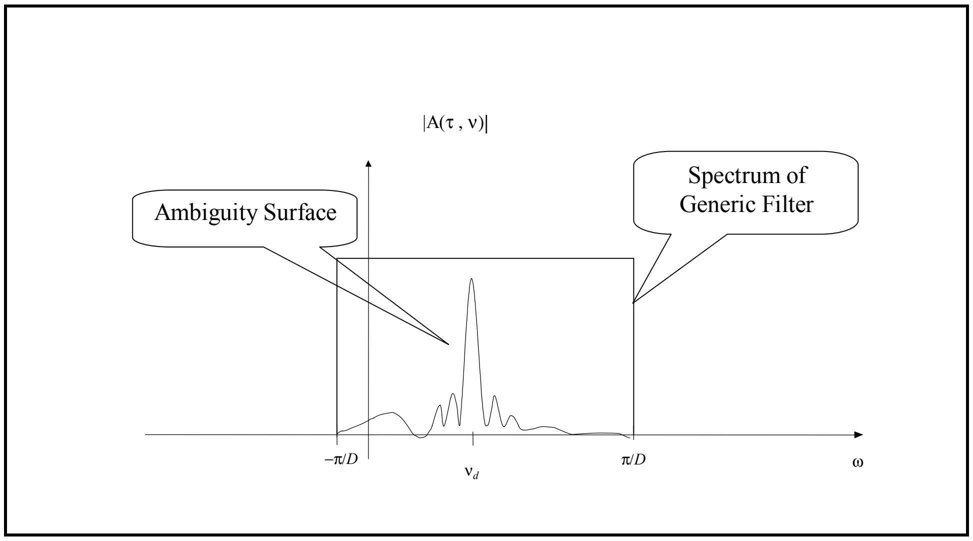

Figure 20 Improved Lag-Product Filtering Using More Appropriate Filter

As can be seen in Figure 20, the filter is specifically designed for the Doppler range of interest, since the ambiguity surface is zero outside of +/- π/D. This minimizes any aliasing that would occur. The decimation rate, D, is used in this instance instead of L because in the "Fine-Mode" algorithm from [1], D was equal to the filter length. Since a more generic filter is being applied, this constraint no longer applies. The decimation rate, D, must satisfy $D \leq L$.

The filter from Figure 20 is an "ideal" filter and is impossible to create because it is non-causal.

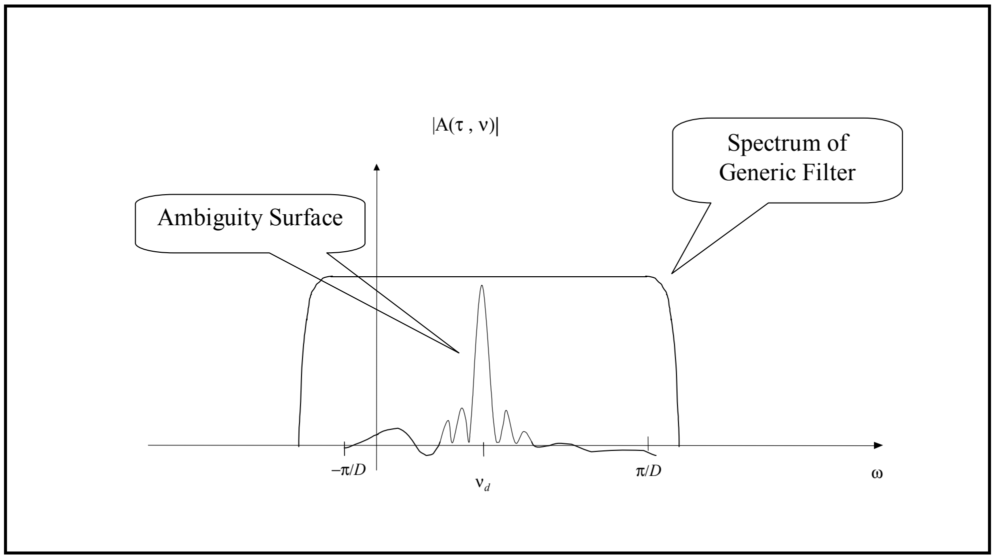

Figure 21 Generic Filter Used In Practice

In practice, an approximation to the rectangular filter such as in Figure 21 is used. As shown in Figure 21, the approximation to the rectangular window has a flat pass-band over the Doppler range of interest and a transition band that extends outside of the Doppler range of interest. This generic filter can be designed to meet any requirements.

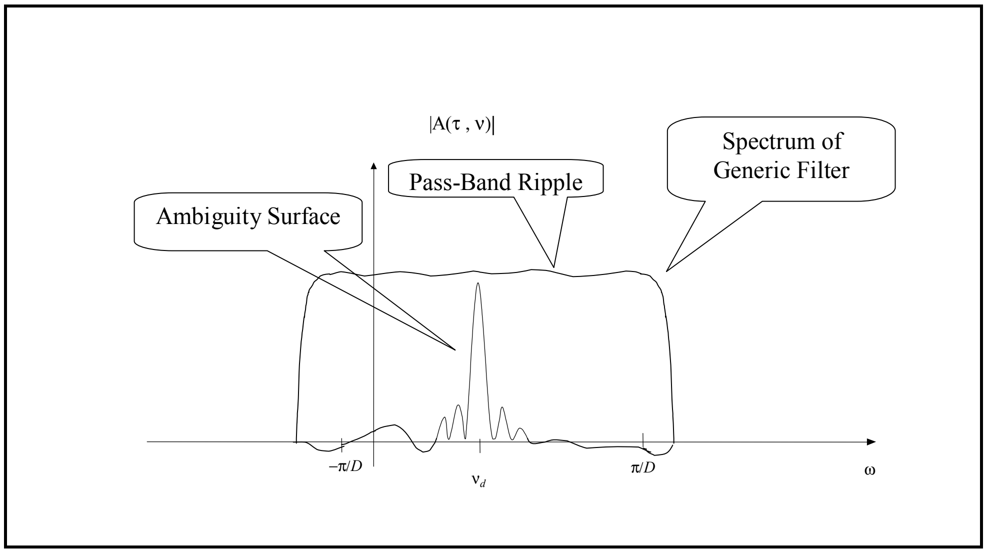

Figure 22 Generic Filter with Pass-Band Ripple

In fact, the requirements on the filter in Figure 21 can be relaxed to include ripple in the pass-band (see Figure 22), be longer in length, and/or have a much different transition band than the one shown in Figure 21. In all of these cases, errors are introduced because of pass-band ripple and/or the "decay" in the transition band. These errors need to be and can be corrected over the specific Doppler range of interest.

To fix these errors, recall from (26) that $A(\tau,\nu)$ is the DTFT of $r_\tau[n]$.

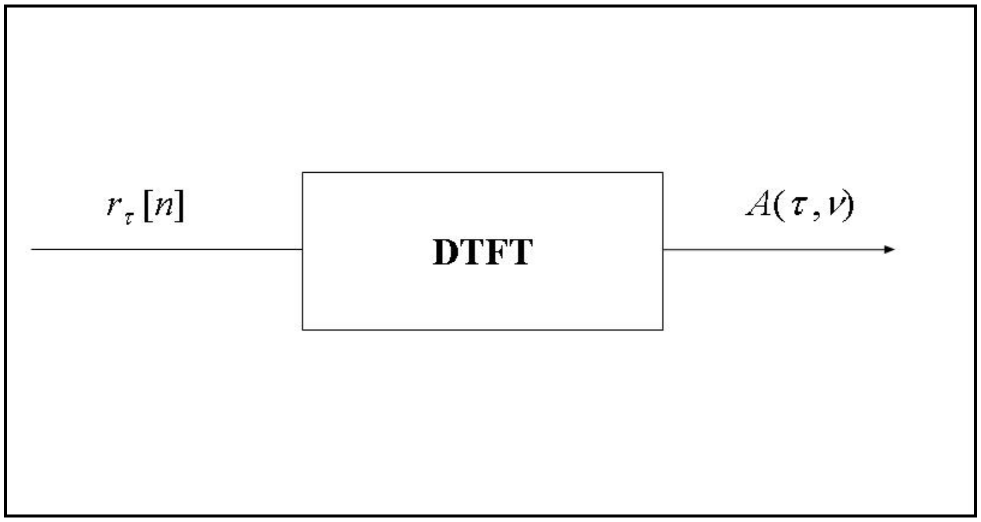

Figure 23 Block Diagram of DTFT of $r_\tau[n]$

To apply the generic filter, $r_\tau[n]$ is convolved with $h[n]$ as shown in Figure 24.

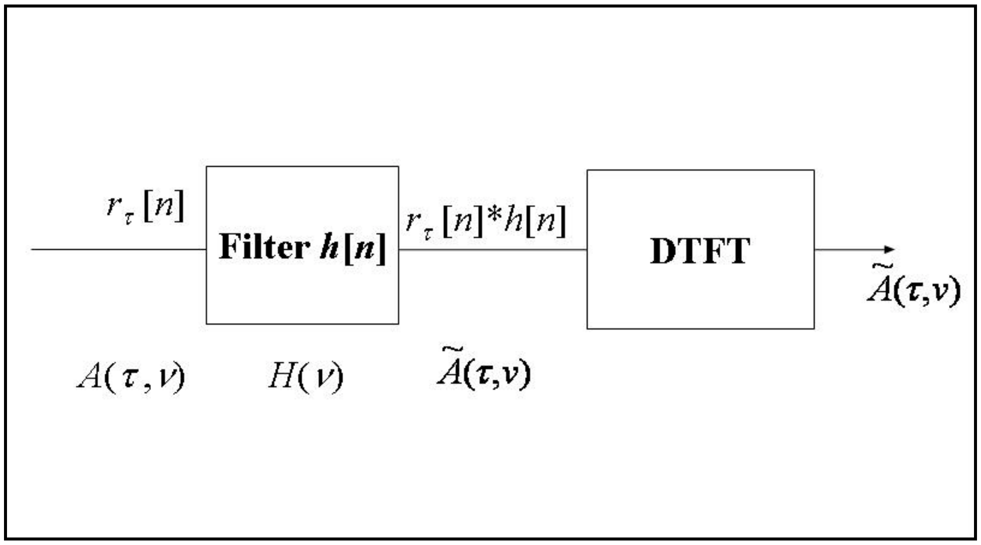

Figure 24 Lag Product Convolution with Generic Filter

After the convolution in the time domain, the result in the frequency domain is

$$\tilde{A}(\tau,v) = H(v)A(\tau,v)\,, \tag{39}$$

where $H(v)$ is the DTFT of $h[n]$ and $A(\tau, v)$ is the DTFT of $r_\tau[n]$. Recall from Figure 21 that $H(v)$ is > 0 over the Doppler range of interest ($[-\pi/D, \pi/D]$). Therefore, over the Doppler range of interest there exists a $1/H(v)$ such that

$$\tilde{\tilde{A}}(\tau,v) = \frac{\tilde{A}(\tau,v)}{H(v)}\,. \tag{40}$$

The result in (40) accounts for and fixes the aliasing error due to the transition band of the filter $h[n]$ extending outside the Doppler range of interest and is a result presented in Section III of [2]. The result here is no different than the result obtained in (38). The lag product ($r_\tau[n]$) is still filtered using a FIR filter and is then decimated by a factor D. However, the filter used is a more generic filter that can be any length, with filters approximating the "ideal" filter being more complex due to the increased number of taps. With the use of the generic filter $h[n]$, the decimation factor is no longer limited to being equal to the filter length; it is now $\leq F_s/\nu_{max}$.

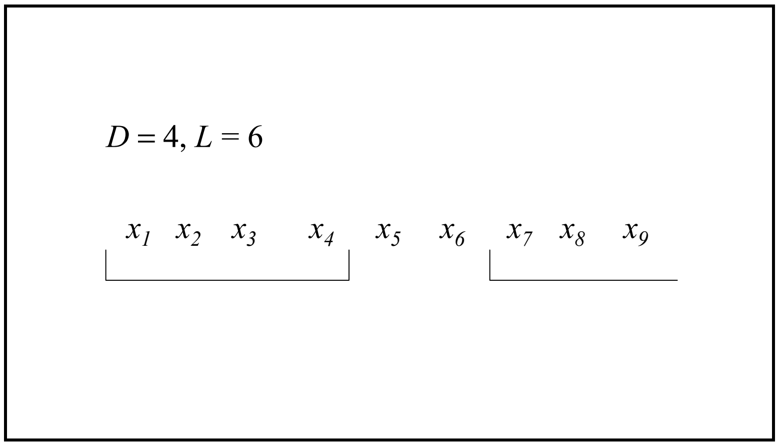

Figure 25 Result of Decimation Factor > Filter Length

If the decimation factor is greater than the filter length, this causes problems in that samples are skipped, as shown in Figure 25 where the filter length is 4 and the decimation factor is 6. Care should be made to assure that the decimation factor is ≤ than the filter length.

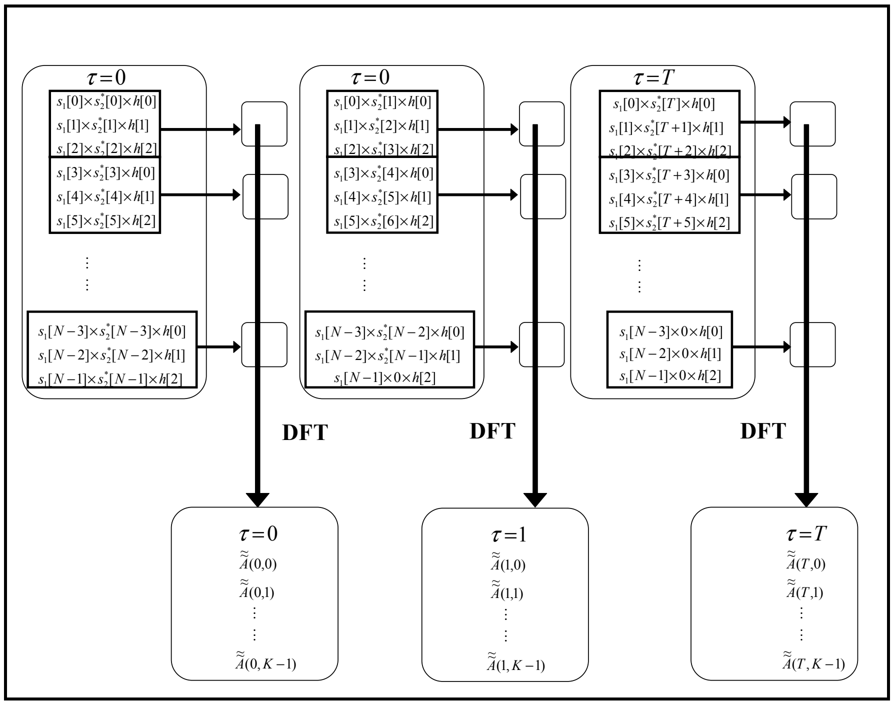
$\tau=0$
$s_1[0]\times s_2^*[0]\times h[0]$
$s_1[1]\times s_2^*[1]\times h[1]$
$s_1[2]\times s_2^*[2]\times h[2]$
$s_1[3]\times s_2^*[3]\times h[0]$
$s_1[4]\times s_2^*[4]\times h[1]$
$s_1[5]\times s_2^*[5]\times h[2]$
$s_1[N-3]\times s_2^*[N-3]\times h[0]$
$s_1[N-2]\times s_2^*[N-2]\times h[1]$
$s_1[N-1]\times s_2^*[N-1]\times h[2]$
$\tau=0$
$s_1[0]\times s_2^*[1]\times h[0]$
$s_1[1]\times s_2^*[2]\times h[1]$
$s_1[2]\times s_2^*[3]\times h[2]$
$s_1[3]\times s_2^*[4]\times h[0]$
$s_1[4]\times s_2^*[5]\times h[1]$
$s_1[5]\times s_2^*[6]\times h[2]$
$s_1[N-3]\times s_2^*[N-2]\times h[0]$
$s_1[N-2]\times s_2^*[N-1]\times h[1]$
$s_1[N-1]\times 0\times h[2]$
$\tau=T$
$s_1[0]\times s_2^*[T]\times h[0]$
$s_1[1]\times s_2^*[T+1]\times h[1]$
$s_1[2]\times s_2^*[T+2]\times h[2]$
$s_1[3]\times s_2^*[T+3]\times h[0]$
$s_1[4]\times s_2^*[T+4]\times h[1]$
$s_1[5]\times s_2^*[T+5]\times h[2]$
$s_1[N-3]\times 0\times h[0]$
$s_1[N-2]\times 0\times h[1]$
$s_1[N-1]\times 0\times h[2]$
DFT
DFT
DFT
$\tau=0$
$\tilde{\tilde{A}}(0,0)$
$\tilde{\tilde{A}}(0,1)$
$\tilde{\tilde{A}}(0,K-1)$
$\tau=1$
$\tilde{\tilde{A}}(1,0)$
$\tilde{\tilde{A}}(1,1)$
$\tilde{\tilde{A}}(1,K-1)$
$\tau=T$
$\tilde{\tilde{A}}(T,0)$
$\tilde{\tilde{A}}(T,1)$
$\tilde{\tilde{A}}(T,K-1)$

Figure 26 Computation of Ambiguity Function Using "Fine-Mode" Generic Filter

Used with Permission From [10]

Procedures:

1. Determine the number of time delays and Doppler shifts of interest.
2. Design a generic filter $h[n]$ based on the number of Doppler shifts of interest. Length of the filter (L) has an upper limit of $F_s/2\pi v_{max}$, where v_{max} is the max Doppler shift frequency.
3. Set the decimation rate $D = L$.
4. Zero-pad $s_1[n]$ and $s_2[n]$ to a length that is divisible by the length of h[n] (L).
5. Determine the number of blocks M = length($s1[n]$)/L.
6. Compute each Inner Sum and take the DFT

```
for $\tau = 0$ to T
  for $m = 0$ to $M-1$
    /* Get data for this block */
    $\tilde{r}_\tau[m] = \sum_{n=mL}^{mL+L-1} r_\tau[n]h[n-mL]$ , where $r_\tau[n] = s_1[n]s_2^*[n+\tau]$ is the lag product

    Zero-pad $\tilde{r}_\tau[m]$ to get appropriate Doppler bin spacing
  end for on $m$

  $A(\tau,k) = \sum_{m=0}^{K-1} \tilde{r}_\tau[m] e^{\frac{-j2\pi kmL}{K}}$ , for $k = 0, 1, 2, \ldots, K$

  /* $A(\tau,k)$ is nothing more than the DFT of the filtered and decimated lag-
   product */
end for on $\tau$
```

Figure 27 Procedures for Computation of Ambiguity Function Using "Fine-Mode"Generic Filter

After filtering the lag-product, the magnitude and phase of the lag-product will be modified. Having a modified phase is undesirable because it distorts true time delay and Doppler calculated off of the ambiguity surface. To correct for the phase delay, the phase delay must be subtracted off from the phase measured off of the ambiguity surface. Therefore, the phase delay has to be known. This makes the generic filter be a linear phase filter for the ability to have a constant phase delay, and from properties of linear phase filters, the generic filter must be symmetric [2]. For a second constraint, recall that the reason for filtering the lag-product was to narrow the Doppler range down to some range of interest, because the original contains unneeded frequencies. Therefore, the generic filter outside of the Doppler range of interest the filter must be zero [2].

To be able to filter the lag-product ($r_\tau[n]$), a discrete time convolution needs to be performed. However, recall that convolution in the discrete time domain gives rise to multiplication in the frequency domain. Instead of performing the convolution in the discrete time domain, it would be more efficient to compute it in the frequency domain. To see this, start with (36) but replace the "all-ones" filter $1[n]$ with the generic filter $h[n]$, where $h[n]$ is defined as

$$h[n] = \begin{cases} arbitrarily, & n = 0, 1, \dots, L-1 \\ 0, & \text{otherwise} \end{cases}, \tag{41}$$

where L is the chosen block length. Since a generic filter is being used, the decimation factor applied after filtering may be ≤ than L. Let the decimation factor be $D < L$. Applying this decimation factor and substituting (41) into (36) the following is obtained

$$A(\tau, v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi vmD}{F_s}} \sum_{n=mD}^{mD+L-1} r_\tau[n] h[n - mD], \tag{42}$$

Substituting (25) into (42), (42) becomes

$$A(\tau, v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi vmD}{F_s}} \sum_{n=mD}^{mD+L-1} s_1[n] s_2^*[n+\tau] h[n - mD]. \tag{43}$$

Rearranging terms in (43) and arbitrarily grouping the $h[n\text{-}m\mathrm{D}]$ term with $s_1[n]$ gives

$$A(\tau,v)=\sum_{m=0}^{M-1} e^{\frac{-j2\pi vmD}{F_s}} \sum_{n=mD}^{mD+L-1} s_1[n]h[n-mD]s_2^*[n+\tau]. \tag{44}$$

Recall the limitation that was placed on (44) where $A(\tau,v)$ had to be zero-padded to ensure it was a length divisible by L. The number of zeros that need to be added to the end of the sequence given the filter length L, decimation factor D, and the number of signal samples N can be verified to be

$$number\ of\ zeros = z(L,D,N) = L + D\left\lceil \frac{N-L}{D} \right\rceil - N, \tag{45}$$

where $\lceil\ \rceil$ means take the ceiling of the division. The number of blocks needed is simply

$$number\ of\ blocks = b(L,D,N) = \left\lceil \frac{N-L}{D} \right\rceil + 1. \tag{46}$$

To efficiently compute the convolution in (44), the computation should be done in the frequency domain, as is discussed in Section 5.6 on page 148 in [9]. In order to do this, arbitrarily group $h[n]$ and $s_1[n]$ together to get

$$f_m[n] = s_1[n]h[n-mD]. \tag{47}$$

Substituting (47) into (44) gives

$$A(\tau, v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi v m D}{F_s}} \sum_{n=mD}^{mD+L-1} f_m[n] s_2^*[n+\tau]. \tag{48}$$

Since it is desired to compute the inner sum of (48), let

$$w_\tau[m] = \sum_{n=mD}^{mD+L-1} f_m[n] s_2^*[n+\tau], \tag{49}$$

where $w_\tau[m]$ is the inner sum of (48). Instead of viewing τ as fixed and m as variable as in (49), the roles can be arbitrarily switched so that m is fixed and τ is variable. This yields a correlation of finite duration signal streams f_m and s_2

$$\widetilde{w}_m[\tau] = \sum_{n=mD}^{mD+L-1} f_m[n] s_2^*[n+\tau]. \tag{50}$$

Note that if the decimation rate D is < the filter length L, there will be overlap between the blocks. Since it has been established that it is more efficient to compute a convolution in the frequency domain, the DFT must be used as is explained in Section 5.6 of [9]. This leads to the result

$$\widetilde{w}_m[\tau] = IDFT\{F_m[k] S_2^*[k]\},\ 0 \le \tau \le T,\ 0 \le k \le 2L-1,\ T \le 2L, \tag{51}$$

where k is DFT indices for both F_m and S_2, and $F_m[k]$ and $S_2^*[k]$ are the DFTs of $f_m[n]$ and $s_2[n+\tau]$, respectively for the appropriate block. The DFT can be used here even though we have a correlation and not a convolution because the both convolution and correlation can be computed in the frequency domain. For correlation in the frequency domain, there is a multiply and a conjugation, whereas for convolution, there is only a multiplication.

The equation in (51) states a lower limit on L. It can be viewed as either T limiting the length of L or as the filter length of L limiting T. The limit is $L \geq T/2$, where T is the maximum number of time delays. This maximum number of time delays works with the maximum number of Doppler shifts to put an upper (see (32)) and lower bound on the filter L, unlike the "Fine-Mode" algorithm in [1] which did not have a lower bound. This puts a constriction on the usage of this method in that it may not be the ideal method for a large number of time delays.

To ensure that the correlation in (50) is linear and is not circular, proper zero-padding must be applied. Since the correlation is done over a filter length L for all blocks, both streams (f_m and s_2) for a particular block will always be length L, therefore the proper zero-padding for every block is $2L$-1, as is discussed in Section 4.7 of [9]. To complete the computation of the ambiguity function, a DFT over the Doppler frequencies of interest must be computed for each block of $\widetilde{w}_m[\tau]$ computed in (51). The result in (51) is the result form the π^3 method from Section III of [2].

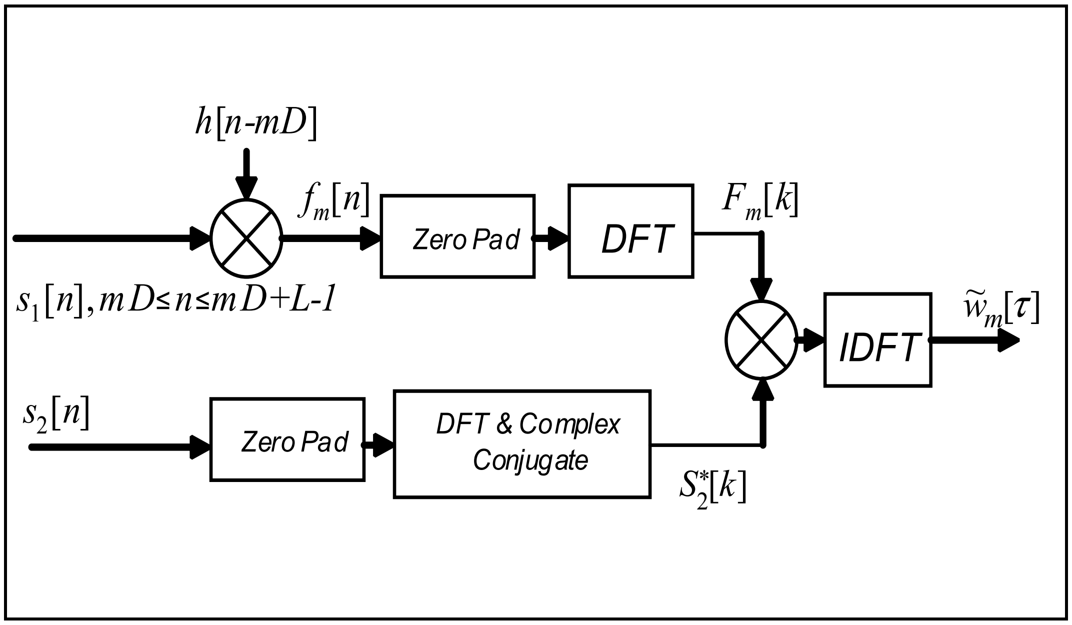

Figure 28 Discrete Time Correlation in Frequency Domain for the *m*th block

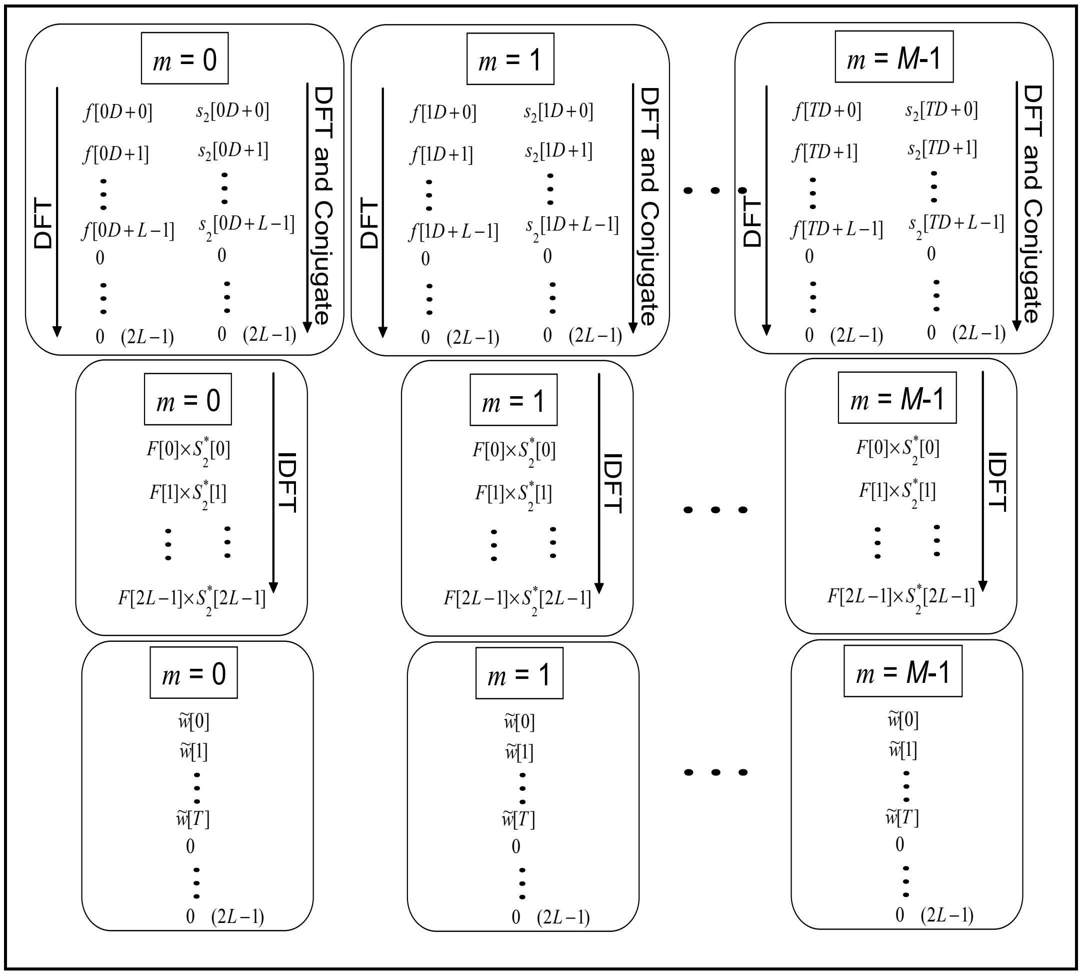

Figure 29 Computation of Ambiguity Function Using "Fine-Mode" Generic Frequency Domain Part I

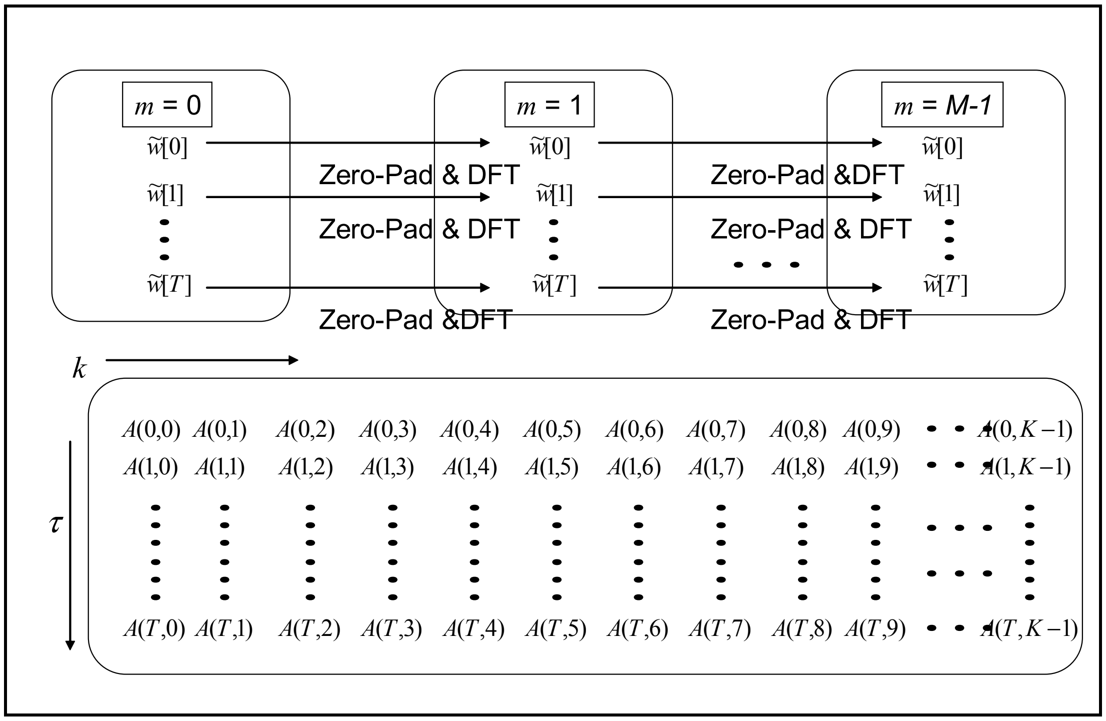
m = 0
m = 1
m = M-1
$\tilde{w}[0]$
$\tilde{w}[1]$
$\tilde{w}[T]$
Zero-Pad & DFT
Zero-Pad &DFT
k
τ
$A(0,0)$ $A(0,1)$ $A(0,2)$ $A(0,3)$ $A(0,4)$ $A(0,5)$ $A(0,6)$ $A(0,7)$ $A(0,8)$ $A(0,9)$ $A(0,K-1)$
$A(1,0)$ $A(1,1)$ $A(1,2)$ $A(1,3)$ $A(1,4)$ $A(1,5)$ $A(1,6)$ $A(1,7)$ $A(1,8)$ $A(1,9)$ $A(1,K-1)$
$A(T,0)$ $A(T,1)$ $A(T,2)$ $A(T,3)$ $A(T,4)$ $A(T,5)$ $A(T,6)$ $A(T,7)$ $A(T,8)$ $A(T,9)$ $A(T,K-1)$

Figure 30 Computation of Ambiguity Function Using "Fine-Mode" Generic Frequency Domain Part II

Procedures:

1. Determine the number of time delays and Doppler shifts of interest.
2. Design the generic filter $h[n]$ based on the number of time delays and Doppler shifts of interest. Length of the filter (L) is bounded by $[T, F_s/2\pi v_{max}]$, where T is the max number of time-delays and v_{max} is the max Doppler shift frequency.
3. Chose the decimation rate $D \le F_s/v_{max}$
4. Zero-pad $s_1[n]$ and $s_2[n]$ to a length that is divisible by the length of $h[n]$ (L).
5. Determine the number of blocks M = length($s_1[n]$)/L.
6. Compute each Inner Sum and take the DFT

```
for $m = 0$ to $M-1$
  /* Get data for this block */
  for $n = 0$ to $L-1$
    $f_m[n] = s_1[n + mD] \times h[n - mD]$
    $s_{2,m}[n] = s_2[n + mD]$
  end for on $n$

  Zero-Pad $f_m[n]$ and $s_{2,m}[n]$ to $2L-1$

  /* Take the DFT of $f_m[n]$ and $s_{2,m}[n]$ */
  $F_m[k] = \sum_{n=0}^{2L-1} f_m[n] e^{\frac{-j2\pi kn}{2L}}$
  $S_{2,m}[k] = \sum_{n}^{2L-1} s_{2,m}[n] e^{\frac{-j2\pi kn}{2L}}$

  /* Multiply $F_m[k]$ and $S_{2,m}[k]$ together */
  $W_m[k] = F_m[k] \times S_{2,m}^*[k]$

  /* Take the IDFT of $W_m[k]$ */
  $\widetilde{w}_m[\tau] = \sum_{k=0}^{2L-1} F_m[k] \times S_{2,m}^*[k], 0 \le \tau \le T, 0 \le k \le 2L-1, T \le 2L$

  Zero pad $\widetilde{w}_m[\tau]$ to ensure correct Doppler bin spacing
end for on $m$

$A(\tau, k) = \sum_{m=0}^{M-1} \widetilde{w}_m[\tau] e^{\frac{-j2\pi kmD}{N}}$
```

Figure 31 Procedures for Computing Ambiguity Function Using "Fine-Mode" Generic Frequency Domain

Another way to reach the result in (51) is to use Parseval's theorem. To see this, first start with the result in (50) and let the proper zero-padding be performed such that linear correlation will be performed. This yields

$$\widetilde{w}_m[\tau] = \sum_{n=mD}^{mD+2L-1} f_m[n] s_2^*[n+\tau]. \tag{52}$$

Using Parseval's theorem, (52) becomes

$$\widetilde{w}_m[\tau] = \sum_{n=mD}^{mD+2L-1} f_m[n] s_2^*[n+\tau] = \frac{1}{2L} \sum_{k=0}^{2L-1} F_m[k] S_2^*[k] e^{\frac{j2\pi k\tau}{2L}} . \tag{53}$$

The result in (53) is nothing more than the IDFT of $F_m[k]\, S_2^*[k]$, thus the result in (53) and (51) are equivalent.

3.2 Filter Bank Viewpoint

Another method for computing the ambiguity function is to start with (24) from Chapter 3 Section 3.1. The result in (24) looks like the linear convolution defined in Section 4.7 of [9], except there is a "+" in s_2 in (24) instead of a "-" sign. Recall that convolution and correlation are identical except that before convolution is performed, there is a "pre-flip". If the data in (24) is "pre-flipped", the result in (24) becomes

$$A(\tau,\nu) = \sum_{n=0}^{N-1} s_1[n] s_2^*[n-\tau] e^{\frac{-j2\pi\nu n}{F_s}} . \tag{54}$$

If s_1 and s_2 are convolved, then s_2 will be flipped about n = 0, thus giving the correlation in (24). Combining, s_1 and the complex exponential from (54) gives

$$A(\tau,\nu) = \sum_{n=0}^{N-1} h_\nu[n] s_2^*[n-\tau], \tag{55}$$

where $h_v[n] = s_1[n]\ e^{\frac{-j2\pi vn}{F_s}}$. The result in (55) is nothing more than a convolution between a signal (s_2) and a FIR filter (s_1), where s_1 are the tap values of the FIR filter that are dependent on the frequencies in v. The s_2 signal is effectively passed through the filter (s_1). This result is same as from Section II of [2]. Although this method is computed by "brute force", it may be suitable for some situations. However, there is no decrease in complexity due because no decimation or approximation is performed.

3.3 2-D DTFT of Cross-Spectra Matrix

Often it is desirable to do frequency domain processing on signals, in order to determine if there is interference that must be removed [11] [1]. This is very important when it comes to computing the ambiguity function because if there is any interference, the TDOA/FDOA measurements measured off the ambiguity surface will not be accurate, thus causing the emitter location to be incorrect. In a precision emitter location system, this yields undesirable performance. Therefore it is desirable to compute the Fourier transform of each signal using a window which allows for spectral analysis, as is shown on page 78 of [8] [11].

The "Fine-Mode" Generic Frequency Domain method discussed in Section III in [2] allows for some spectral analysis, but only for one signal. The filter used for that method can be thought of as a window applied to signal blocks of one signal stream prior to computing the DFT(see (47) - (51)) [11]. What is desired is to find a balance between requirements placed on the window (spectral analysis) and the decimation requirements (decimation factor D). The first step in this analysis is to start with (24) and zero pad s_1 and s_2 appropriately such that the sum in (24) can be broken up into M non-overlapping blocks each of length L as was done in (27). Recall that $ML \geq N$ and $L << N$. Substituting (25) into (27) and introducing a change of variable in the inner summation produces

$$A(\tau,\nu) = \sum_{m=0}^{M-1} \sum_{p=mL}^{mL+L-1} s_1[p] s_2^*[p+\tau] e^{\frac{-j2\pi p\nu}{F_s}}. \tag{56}$$

Now extend the *M* signal blocks of length *L* in length *N* signals as follows

$$s_{1,m}[n] = \begin{cases} s_1[n], & \mathit{for}\ n = mL, mL+1, \dots, mL+L-1 \\ 0, & \text{otherwise} \end{cases}$$
$$s_{2,m}[n] = \begin{cases} s_2[n], & \mathit{for}\ n = mL, mL+1, \dots, mL+L-1 \\ 0, & \text{otherwise} \end{cases}, \tag{57}$$

where *m* is the block index from (56) and *n* goes from 0 to *N*-1. Using the signal blocks from (57) allows an approximation for the $A(\tau,v)$ to be made

$$A(\tau,v) \approx \widetilde{A}(\tau,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} s_{1,m}[n]s_{2,m}^{*}[n+\tau]e^{\frac{-j2\pi nv}{F_s}} . \tag{58}$$

The approximation (58) results from the fact that the non-overlapping signal data blocks have gaps in the two signal streams *after* they are multiplied together. This is different from (56), where no gaps in the signal data streams exist after they are multiplied. Another way to view this is that in (56) a *global* shift scheme is used to impart the specific time delay, τ, on the s_2 signal by zeros into a block and signal samples from that block to the next block (block-to-block shifting). In (58) a *local* shift scheme is used based on the partitioning that was defined in (57) such that no block-to-block shifting is performed to impart the time delay. To impart the time delay, τ, in (58), τ zeros replace τ signal samples in the block of *L* signal samples. Thus as the time delay becomes larger and larger, less and less data is correlated because data is being lost as zeros are being inserted . To minimize the impact of this “block-shift loss”, the maximum time delay should satisfy $T = \max(|\tau|) << N$ [11]. A method will be discussed later that will remove this impact.

As noted above, for each block, *m*, there will be at most *L* non-zero signal samples. Therefore the inner sum in (58) can be re-written as

$$\sum_{n=0}^{N-1} s_{1,m}[n]s_{2,m}^{*}[n+\tau]e^{\frac{-j2\pi nv}{F_s}} = e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{L-1} \widetilde{s}_{1,m}[n]\widetilde{s}_{2,m}^{*}[n+\tau]e^{\frac{-j2\pi nv}{F_s}} \quad , \tag{59}$$

where $\widetilde{s}_{1,m}[n]$ and $\widetilde{s}_{2,m}[n]$ are obtained from circular shifting $s_{1,m}$[n] and $s_{2,m}$[n] to the left by mL samples. The complex exponential out in front of the right-hand side of (59) is due to the effect of a circular shift on the DTFT [9]. Substituting (59) into (58) yields

$$A(\tau,v) \approx \widetilde{A}(\tau,v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{L-1} \widetilde{s}_{1,m}[n]\widetilde{s}_{2,m}^{*}[n+\tau]e^{\frac{-j2\pi nv}{F_s}} \quad . \tag{60}$$

Applying the narrowband approximation to (60) as was discussed in (29) - (32) gives

$$A(\tau,v) \approx \widetilde{A}(\tau,v) \approx \widetilde{\widetilde{A}}(\tau,v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{L-1} \widetilde{s}_{1,m}[n]\widetilde{s}_{2,m}^{*}[n+\tau], . \tag{61}$$

where the inner sum in (61) is similar to the "Fine-Mode" method in [1], except that (61) has the "block-shift loss" discussed above. And as discussed in (32) - (33) for the "Fine-Mode" method, because of the use of the narrowband approximation, an upper bound on L (filter length/decimation rate) is established such that $L << F_s/2\pi v_{max}$ [11].

Using Parseval's Theorem, the inner-sum on the right-hand side of (61) can be re-written as

$$\sum_{n=0}^{L-1} \widetilde{s}_{1,m}[n]\widetilde{s}_{2,m}^{*}[n+\tau] = \frac{1}{2L} \sum_{k=0}^{2L-1} \widetilde{S}_{1,m}[k]\widetilde{S}_{2,m}^{*}[k]e^{\frac{j2\pi\tau k}{2L}} \ , 0 \le \tau \le T \le 2L \, , \tag{62}$$

where $\widetilde{S}_{1,m}[k] = DFT\{\widetilde{s}_{1,m}[n]\}$ and $\widetilde{S}_{2,m}[k] = DFT\{\widetilde{s}_{2,m}[n]\}$ have both been zero-padded to the appropriate length to ensure that the correlation on the left-hand side of the equation in (62) is linear and not circular. As mentioned for the "Fine-Mode Generic Frequency Domain method" a lower bound is placed on *L*. The limit is $L \geq T/2$, where *T* is the maximum number of time delays [11]. Note that the right hand side of the equation in (62)is nothing more than a 2*L*-point inverse DFT.

Substituting (62) into (61) gives

$$\widetilde{\widetilde{A}}(\tau,v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \left[\frac{1}{2L} \sum_{k=0}^{2L-1} \widetilde{S}_{1,m}[k]\widetilde{S}^*_{2,m}[k] e^{\frac{j2\pi\tau k}{2L}} \right], \tag{63}$$

where the outside sum is a DTFT and the inside sum is an inverse DFT. Since the product $\widetilde{S}_{1,m}[k]\widetilde{S}^*_{2,m}[k]$ can be viewed as a cross-spectrum, the approximate ambiguity function in (63) will be referred to as the "non-windowed cross-spectrum" (CS) method [11]. Note that the DTFT in (63) can be computed on a grid using a DFT via the FFT algorithm, making sure the appropriate zero-padding is produced to get the desired Doppler bin spacing [11].

An *approximate* CS method has been shown, but the "block-shift loss" is a serious problem. This problem is now addressed. As was shown in (34) - (38), the inner sum of (63) can be written in the time domain as a correlation of a decimating filter (h_τ[n]) and a lag product signal (r_τ[n] = $s_1[n]s_2^*[n+\tau]$), which is given by

$$\widetilde{\widetilde{A}}(\tau,v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{2L-1} h_\tau[n] r_\tau[n+mL], \tag{64}$$

where to get the CS method for $\tau \geq 0$ select

$$h_\tau[n] = \begin{cases} 0, & for\ n = 0, 1, 2, \ldots, \tau - 1 \\ 1, & for\ n = \tau, \tau+1, \ldots, L-1 \\ 0, & \text{otherwise} \end{cases}, \tag{65}$$

which depends on the time delay value τ [11]. A similar case can be made for $\tau \leq 0$. Note that zero-padding to $2L$ has been performed to ensure linear correlation. The decimation filter h_τ[n], suffers from the "block-shift loss" discussed above. As the time delay becomes larger and larger, gaps are formed as the more and more zeros replace the ones in the filter. After decimation, the corresponding frequency response of the filter ($H_\tau[f]$) gets aliased back into the Doppler region of interest. For $\tau = 0$, the nulls of the aliased versions of $H_\tau[f]$ line up at the origin as shown in Figure 19. When $\tau \neq 0$, the length of the filter h_τ[n] decreases, and this causes the nulls of the aliased versions of $H_\tau[f]$ not line up, as is shown in Figure 32, which is same example used to create Figure 18 - Figure 19 only a time delay of 10 has been used, which results in a shorter filter in (65) (length of 90 for delay of 10).

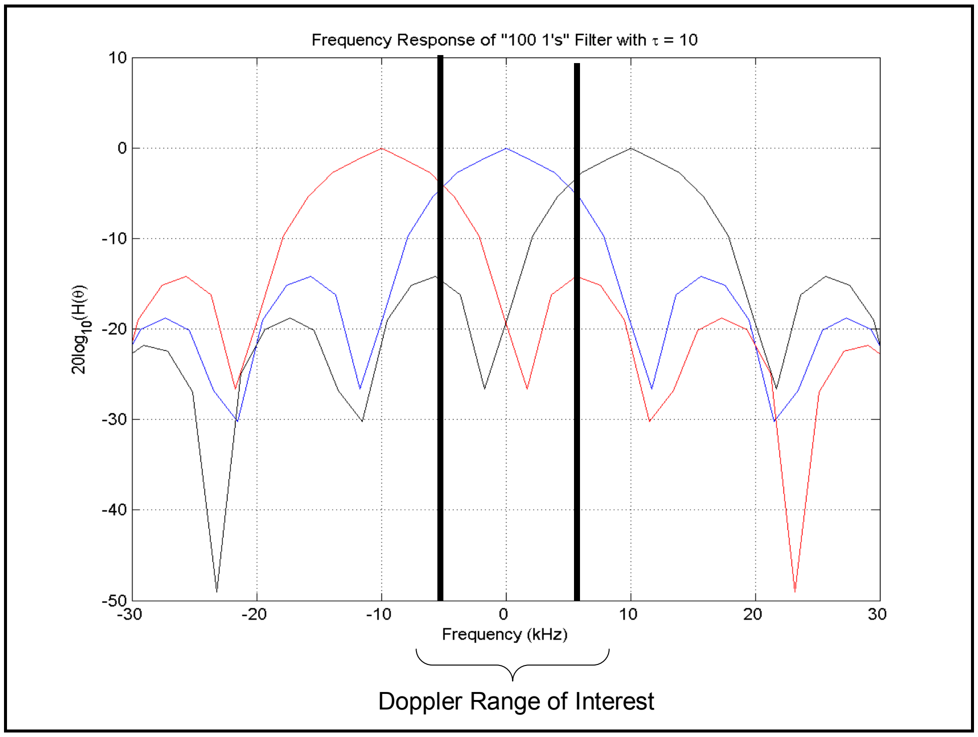

Figure 32 Aliasing of Decimation Filter for $\tau = 10$

Used with Permission from [11]

This aliasing error becomes worse as the time delay ($|\tau|$) increases. Thus, the CS method in (63) gets worse for large time delay values [11]. This is different from the "Fine-Mode" method from [1], such that the decimation filter defined in (35) doesn't depend on τ[11]. As shown in Figure 19, the aliased version of the frequency response of the filter have nulls at the center of the Doppler range of interest, namely zero Doppler. This minimizes the impact of the aliasing [11].

The aliasing problem due to the "block-shift loss" in the CS method can be removed by appropriately choosing window functions for both signal streams and a decimation rate [11]. Start with (63) and replace the non-window DFTs $\widetilde{S}_{1,m}[k]$ and $\widetilde{S}^*_{2,m}[k]$ with windowed versions $\widetilde{S}_{1,m,w}[k]$ and $\widetilde{S}^*_{2,m,w}[k]$ using a common window $w[n]$ [11]. This makes the time-domain form of (61) become

$$\begin{aligned}\widetilde{\widetilde{A}}(\tau,v) &= \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{2L-1} (w[n]\widetilde{s}_{1,m}[n])(w[n+\tau]\widetilde{s}^*_{2,m}[n+\tau]) \\ &= \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{2L-1} h_\tau[n]\widetilde{s}_{1,m}[n]\widetilde{s}^*_{2,m}[n+\tau],\end{aligned} \quad , \tag{66}$$

where $h_\tau[n] = w[n]w[n+\tau]$ is a decimation filter. Note that the inner-sum in (66) has appropriate zero-padding to ensure linear correlation. As was true for the non-windowed case, the decimation filter is different for each value of τ[11]. Re-writing (66) as a correlation of the decimation filter (h_τ[n]) and a lag product signal (r_τ[n] = $s_1[n]s^*_2[n+\tau]$) yields

$$\widetilde{\widetilde{A}}(\tau,v) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{2L-1} h_\tau[n]r_\tau[n+mL] \text{ [11]}. \tag{67}$$

Note that the inner-sum in (67) shows appropriate zero-padding to ensure linear correlation. From (67) it can be seen that there are two requirements for choosing the window $w[n]$: (*i*) $w[n]$ should be chosen such that the resulting decimation filter (h_τ[n]) provides the necessary suppression of aliasing in the Doppler range of interest and (*ii*) it should be chosen such that spectra ($\widetilde{S}_{1,m,w}[k]$ and $\widetilde{S}^*_{2,m,w}[k]$) can have appropriate spectral analysis pre-processing (e.g. removal of interferers, selection between co-channel signals, etc…) as is displayed in [8][11]. If the first part is dealt with, it is clear that a non-rectangular window would not work because of the fact the non-rectangular window has a frequency response with a wider main lobe as is shown in [8][11]. This would cause aliased nulls to be even further from zero Doppler. To solve this aliasing problem the time delay dependence must be removed from the filter so that the aliased nulls align at 0 Doppler as is shown in Figure 19. This is not enough, however, because the window must be chosen such that the impact of aliasing due to the “block-shift loss” is minimized [11]. To do this the assumption made above that the two window functions can be the same must be modified to allow for the two window functions to be different. Define $h_\tau[n] = w_1[n]w_2[n+\tau]$. The goal is to choose the two windows such that their product is not dependent on the time delay (τ) [11]. This can be accomplished, as is shown in Figure 34, by having the w_2 window be chosen to be longer than the w_1 window and have a constant value over a range such that the w_1 window can fit inside the w_2 window and can be slid along this constant range. This allows the w_1 window to have any arbitrary shape, but it must be short enough such that it fits inside the w_2 window [11]. Letting the w_2 window be the Tukey window defined in [8] allows for the product of the w_2 and w1 windows (Figure 34) to be independent of the time delay, τ. This also allows the decimating filter to be equal to the w_1 window (shorter window) [11].

Taking the DFTs of $w_1[n]\tilde{s}_{1,m}[n]$ and $w_2[n]\tilde{s}_{2,m}[n]$ we get $\tilde{S}_{1,m,w_1}[n]$ and $\tilde{S}_{2,m,w_2}[n]$, and if we substitute them into (63) the equivalent time-domain form becomes

$$\begin{aligned}\tilde{\tilde{A}}(\tau,v) &= \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{2Lw_2-1} w_1[n]\tilde{s}_{1,m}[n]w_2[n+\tau]\tilde{s}_{2,m}^*[n+\tau] \\ &= \sum_{m=0}^{M-1} e^{\frac{-j2\pi mLv}{F_s}} \sum_{n=0}^{2Lw_2-1} w_1[n]\tilde{s}_{1,m}[n]\tilde{s}_{2,m}^*[n+\tau]\end{aligned} \quad . \tag{68}$$

Note that the inner-sums of (68) have appropriate zero-padding to ensure linear correlation. The shorter window now defines the performance of the decimation filter. The aliasing error is independent of the time delay, τ. However the nulls of the aliased versions of the filter still do not line up at zero Doppler

[11]. To reduce the aliasing error, the decimation rate, D, must be changed to be D < L. This results in computing the various spectra using overlapping data $\tilde{s}_{1,m,o}[n]$ and $\tilde{s}_{2,m,o}[n]$ which are defined as

$$\begin{aligned} \tilde{s}_{1,m,o}[n] &= s_1[mD+n] \quad for\ n = 0,1,2,\ \ldots, L_{w_1}-1 \\ \tilde{s}_{2,m,o}[n] &= s_2[mD+n] \quad for\ n = 0,1,2,\ \ldots, L_{w_2}-1 \end{aligned}, \tag{69}$$

where m is the block index [11]. Substituting (69) into (63) and replacing the decimation rate in the complex exponential outside the "[ ]" with the new decimation rate D gives the windowed CS method

$$\tilde{\tilde{A}}(\tau,\nu) = \sum_{m=0}^{M-1} e^{\frac{-j2\pi mD\nu}{F_s}} \left[\frac{1}{2L} \sum_{k=0}^{2L-1} \tilde{S}_{1,m,o,w_1}[k] \tilde{S}^*_{2,m,o,w_2}[k] e^{\frac{j2\pi\tau k}{2L}} \right]. \tag{70}$$

Note that the inner summation reflects that zero-padding has been performed such that linear correlation is performed from the inverse DFTs in the "[ ]". It can be found that the value of D = 0.88L gives the best alignment of nulls at zero Doppler, thus reducing the impact of aliasing [11]. The result in (70) is implemented by forming a matrix as follows

$$CS[m,k] = \tilde{S}_{1,m,o,w_1}[k] \times \tilde{S}_{2,m,o,w_2}[k], \tag{71}$$

where m is the block index and k is the DFT index from (70). Next zero-pad the columns of the CS matrix to the correct Doppler bin spacing and take the DFT across the columns of the CS matrix to produce the following

$$A_{Doppler}(k,\nu) = \sum_{m=0}^{M-1} CS[k,m] e^{\frac{-j2\pi\nu mD}{F_s}} . \tag{72}$$

Next zero-pad the rows of the $A_{Doppler}(k,\nu)$ matrix to get the correct Doppler bin spacing and take the DFT across the rows of the $A_{Doppler}(k,\nu)$ matrix to get

$$\tilde{\tilde{A}}(\tau,v)=\frac{1}{K}\sum_{k=0}^{K-1}A_{Doppler}(k,v)e^{\frac{j2\pi k\tau}{K}}\ . \qquad (73)$$

The result in (73) is the result described in [5].

Procedures:

1. Determine the number of time delays and Doppler range of interest
2. Design window $w_1[n]$. The shape of the window can be arbitrarily chosen. The window length $L_{w_1} = L$ with an appropriately number of zeros preceding and trailing the window to give it a total length of L_{w2}.
3. The window $w_2[n]$ is a P percent Tukey window (e.g. $P = 75$ % specifies that the center 75% of the window's samples are ones) of length $L_{w2} = 100(L + 2T)/P$, where $T = \max\{|\tau|\}$. Set the decimation rate $D = 0.88L$.
4. Zero-pad $s_1[n]$ and $s_2[n]$ to be divisible by a length L_{w2}.
5. Determine the number of blocks $M = \left\lceil \frac{N - L_{w2}}{D} \right\rceil$.
6. Compute each Inner Sum and take the DFT

```
for $m = 0$ to $M-1$
  /* Form the m^th signal blocks */
  for $n = 0$ to $L_{w2}-1$
    $\tilde{s}_{1,m,o}[n] = s_1[n + mD]$
    $\tilde{s}_{2,m,o}[n] = s_2[n + mD]$
  end for on $n$

  Zero-Pad $w_1[n]\,\tilde{s}_{1,m,o}[n]$ and $w_2[n]\,\tilde{s}_{2,m,o}[n]$ to $2L_{w2}-1$

  /* Take the DFT of $w_1[n]\,\tilde{s}_{1,m,o}[n]$ and $w_2[n]\,\tilde{s}_{2,m,o}[n]$ */
  $\tilde{S}_{1,m,o,w_1}[k] = \sum_{n=0}^{2L_{w2}-1} w_1[n]\tilde{s}_{1,m,o}[n] e^{\frac{-j2\pi kn}{2L_{w2}}}$
  $\tilde{S}_{2,m,o,w_2}[k] = \sum_{n=0}^{2L_{w2}-1} w_2[n]\tilde{s}_{2,m,o}[n] e^{\frac{-j2\pi kn}{2L_{w2}}}$

  /* Multiply $\tilde{S}_{1,m,o,w_1}[k]$ and $\tilde{S}^*_{2,m,o,w_2}[k]$ together to create the */
  /* cross-spectra matrix $CS[m,k]$ whose columns are the spectra of the blocks */
  $CS[m,k] = \tilde{S}_{1,m,o,w_1}[k] \times \tilde{S}^*_{2,m,o,w_2}[k]$
end for on $m$

Zero pad the columns of $CS[m,k]$ to ensure correct Doppler bin spacing

/* Computing the DTFT of the columns of $CS[m,k]$ at the desired Doppler values */
$A_{Doppler}(k,v) = \sum_{m=0}^{M-1} CS[k,m] e^{\frac{-j2\pi vm}{F_s}}$
Zero pad the rows of $A_{Doppler}[k,v]$ to ensure correct Tau bin spacing
/* Compute the IDFT of the rows of $A_{Doppler}(k,v)$ at the desired Tau bin values */
$\tilde{\tilde{A}}(\tau,v) = \frac{1}{K}\sum_{k=0}^{K-1} A_{Doppler}(k,v) e^{\frac{j2\pi k\tau}{K}}$
```

Figure 33 Procedures for Computing 2-D Windowed Cross-Spectrum Method Used with Permisson from [11]

The frequency domain approach outlined in Figure 33 provides a way to remove interfering signals, including ones that are not resolved in time delay or Doppler [5]. Interference removal is designed into the approach using (72). When no noise or interference is present, there is part of the spectrum that runs parallel to the frequency axis and is located at the Doppler shift value between the signals (s_1, s_2). If the phase values along this part of the spectrum are extracted, unwrapped, and plotted, a straight line with a slope equal to the delay between the two signals is the result [5][11]. The key is to take the difference between adjacent Doppler bins for this particular Doppler frequency. This yields a constant value proportional to the time delay [11]. If interfering signals are present, then there will be discontinuities in the phase plot. Given some a priori knowledge about the desired signal's characteristics (bandwidth, frequency, etc…), it is possible to devise an algorithm to remove the frequencies belonging to the interfering signals and retain the desired signal of interest [5][11].

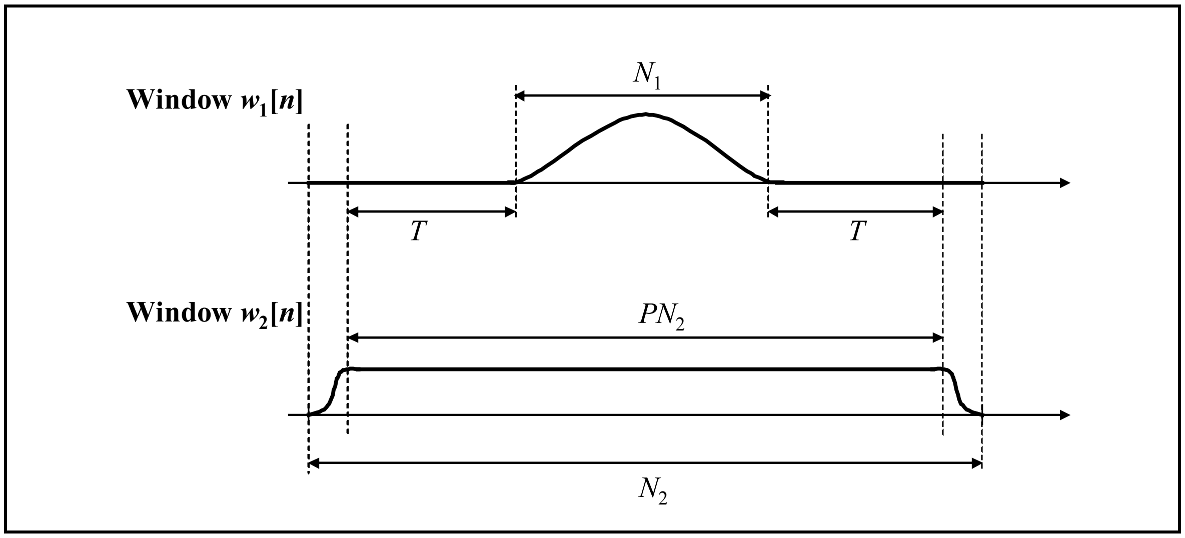

Figure 34: Windows for the two-window version

Chapter 4 Summary of Methods & Computational Complexity

The preceding chapter developed several methods in detail. This chapter provides a summary of the resulting methods in a way that clearly outlines the various practical methods worthy of consideration for practical use. In addition, computation counts are determined for each of the methods.

4.1 Comparison of Methods

The methods discussed in Chapter 3 are all related. Starting with the DTFT form of the ambiguity function as is in (24) one can move directly to using the filter bank method as shown in (55). This method is a “brute-force” direct computation method. This method may be the one of the easiest to compute, but in some applications it may be the more computationally inefficient because no decimation is performed and the FIR filter in this method becomes very large for a large number of Doppler frequencies.

Taking the DTFT form of the ambiguity function in (24), forming the lag-product, and breaking the large sum into smaller non-overlapping sums (with the lengths of the smaller sums equal to the decimation rate) gives the result in (28). Making the approximation in (32) that the speed of light constant is much larger in respect to the velocity of the entity collecting the emitter signal data produces the result in (34). The result in (34) is Seymour Stein’s “Fine-Mode” Method and is less computationally complex than the filter band method because there is decimation included in the algorithm [1]. The decimation is equal to the length of the smaller non-overlapping sums. Generally, the use of data reduction methods (decimation) results in a more computational efficiency.

Replacing the “all-ones” filter from Stein’s “Fine-Mode” method with a more generic filter gains the advantage that the more generic filter can be designed to achieve better Doppler aliasing suppression. The “all-ones” filter allows for aliasing as shown in Figure 19. The argument made by Stein in [1] is that for most applications, this aliasing is acceptable. For some applications, suppression of Doppler aliasing may lead to better TDOA/FDOA accuracy.

The tradeoff in using the generic filter over the “all-ones” filter is that the decimation factor is lower than in Stein’s method. Also because of the use of a more generic filter, the simplicity of the “all-ones” filter is lost. Using the “all-ones” filter did not require any additional multiplies. Using the more generic filter does. This method, therefore, does not have as much complexity reduction as Stein’s method. The “Fine-Mode” Generic method allows for non-overlapping smaller sums, and is a method proposed by Tolimieri and Winograd in [2] and is shown in (40). This method is essentially Stein’s “Fine-Mode” method but with a more generic filter. It also contains corrections for the generic filter used (transition band, pass-band ripple, etc…).

As noticed by Tolimieri and Winograd in [2], it is computationally efficient in many applications to compute the ambiguity function via the frequency domain. This is easily accomplished by breaking up the lag product from Stein’s “Fine-Mode” method [1] or Tolimieri and Winograd’s “Fine-Mode” Generic method [2] into their respective separate signal streams. These signal streams are then broken up further into an integer multiple of overlapping blocks. For the first signal stream, take the DTFT and then apply a window for each block, whereas for the second stream, just take the DTFT for each block. These two streams of block DTFTs are multiplied together on a block by block basis followed by applying an IDTFT on a block by block basis. This method is the “Fine-Mode” Generic Frequency Domain method and is similar to the 2-D Cross Spectra method discussed in [5]and [11]. In the 2-D Cross Spectra method, the second signal stream is windowed in addition to the first. The window on the second signal stream is a special window called the Tukey window. It allows for the first windowed stream to be slid back and forth inside the length of the second windowed stream without changing the lag-product structure. Both frequency domain methods (“Fine-Mode” Generic Frequency Domain and 2-D Cross Spectra), but the 2-D Cross Spectra method in [5] may be preferable because it provides windowed spectral analysis on both signal streams to further aid in excision of interferers.

Figure 35 depicts the relationships between each of the methods presented in Chapter 3.

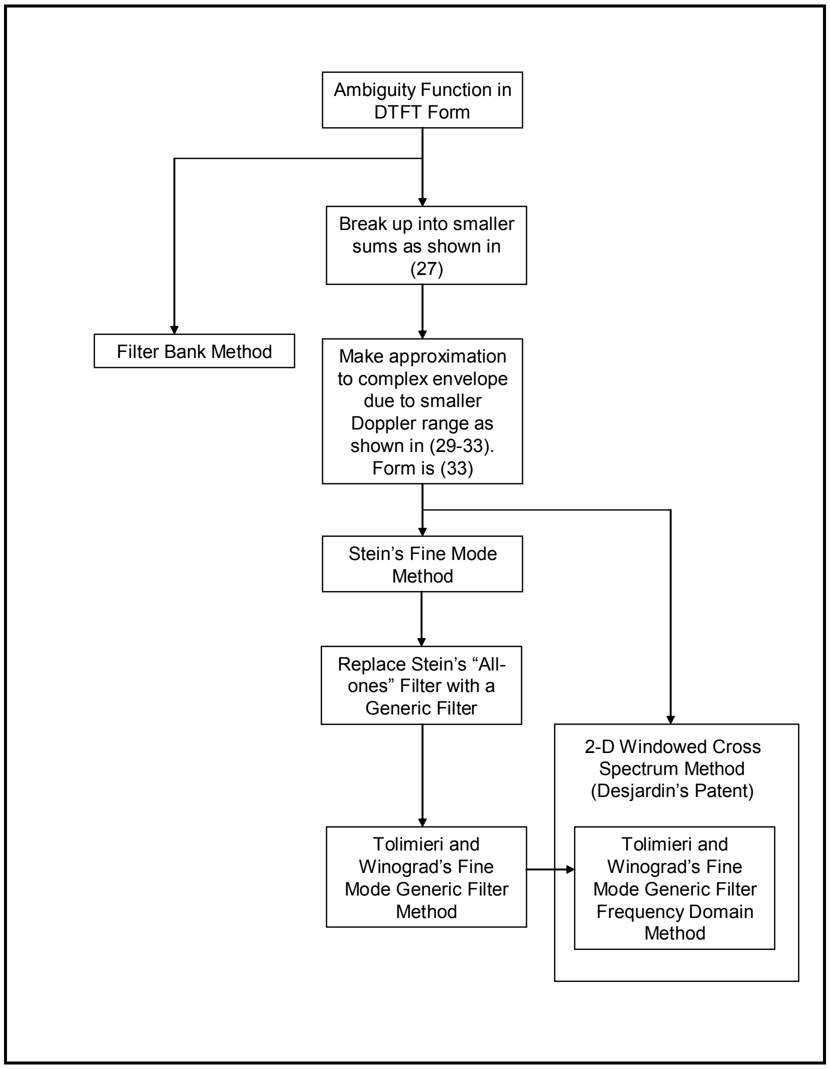
Ambiguity Function in DTFT Form
Break up into smaller sums as shown in (27)
Filter Bank Method
Make approximation to complex envelope due to smaller Doppler range as shown in (29-33). Form is (33)
Stein's Fine Mode Method
Replace Stein's "All-ones" Filter with a Generic Filter
2-D Windowed Cross Spectrum Method (Desjardin's Patent)
Tolimieri and Winograd's Fine Mode Generic Filter Method
Tolimieri and Winograd's Fine Mode Generic Filter Frequency Domain Method

Figure 35 Relationship of Each Method

4.2 Test Setup

In order to compare the pros and cons of each of the various methods a set of tests that would stress the algorithms needed to be developed. This was not an easy task, and the test set that was developed is not designed to test every scenario, but it does provide representative cases to help determine which methods are best suited for certain conditions.

In order to develop the test set, the different items affecting the algorithms namely, frequency, bandwidth, sampling rate, number of Dopplers and time-delays, and decimation factor were examined.

When examining the frequency, the assumption was made that the emitter was stationary and the collection platforms were airborne and would always be traveling at a constant velocity. The value of 250 m/s was chosen for the velocity because a collection platform traveling at 250 m/s is representative of the velocity of many airplanes. Using this assumption that the velocity is always constant (250 m/s) makes the Doppler shift frequency equation in (29) become

$$f = \pm f_o 8.3391 \times 10^{-7}, \tag{74}$$

where the speed of light constant used was c = 299792458 m/s. It is clear from (74) that as the frequency of the signal of interest (f_o) increases, the expected Doppler shift on the signal also increases. If the expected Doppler shift on the signal of interest increases, the Doppler range of interest also increases, thus requiring more Doppler frequencies to be used in the calculation of the ambiguity surface. Therefore in order to have tests where there is variation in the number of Dopplers, three unique frequencies, one from the HF band (5 MHz), one from the VHF band (50 MHz) and one from the UHF band (500 MHz) were chosen. The frequencies in these bands were chosen arbitrarily. Using these three frequencies gives maximum expected Doppler shifts of ±4.169551 Hz, ±41.69551 Hz, and ±416.9551 Hz for the HF band, VHF band, and UHF band respectively. The Doppler shift values that were applied for each different frequency were chosen to be 3 Hz for the 5 MHz frequency, 16 Hz for the 50 MHz frequency, and -322 Hz for the 500 MHz frequency. For simplicity of simulation, the time-delay was set to be 0 nanoseconds for all the test cases.

The next item examined was the bandwidth. As the bandwidth of a signal increases, the rate that the collected signal is sampled must increase in order to avoid the effects of aliasing, as according to Nyquist. The assumption was made for the purpose of simplifying the test set that each of the three frequencies would each have a constant time-bandwidth product (BT) that would produce an approximate 40 dB gain (BT = 10000) in processing. A value of 10000 was chosen because a 40 dB processing gain is a reasonable amount of gain to have for narrow-band correlation processing. Having a constant BT provides for a more realistic simulation where signals at wider bandwidths are collected for shorter times than narrower bandwidths in order to achieve the desired processing gain. Again for simplicity no noise was added to the test signals.

Three different bandwidths were used for the three different frequencies. These bandwidths were chosen based on values shown to be appropriate given the frequencies used as stated at the United States Federal Communications Commission's website [12]. The narrow, medium, and wide bandwidths selected for the 5 MHz, 50 MHz, and 500 MHz frequencies were 5 kHz, 10 kHz, and 20 kHz, respectively. The corresponding sampling rates for the bandwidths were chosen based on the assumption that the data to be correlated would be in complex envelope form. This would allow critical sampling at the actual bandwidth, as per Nyquist. However, each sampling rate was arbitrarily increased by 10 % in order to reduce the inclusion of aliasing errors when critically sampling and using decimation. This gave sampling rates of 5.5 kHz, 11 kHz, and 22 kHz for each of the frequency bands. The sampling rates for the three frequencies and corresponding bandwidths are listed in Table 1.

The next items examined were the number of time-delays and Dopplers. As mentioned in Chapter 3 Section 3.1, the "Fine-Mode" Generic Frequency Domain method has a limit on the number of time-delays. From Figure 35, it can be inferred that the 2-D Cross Spectra method also has this limitation. To compare the performance of these two methods to the others, the number of time-delays or Taus must vary across some range. Because the purpose of this test simulation is to see the accuracy vs. the computational complexity for particular, there is no need to tie an algorithm to a particular geometry, so the Taus may be arbitrarily chosen and reasonably show what happens to accuracy and complex complexity. However, during preliminary testing, and after reviewing the algorithms, it became apparent that certain algorithms provide an inefficient number of Taus that cannot be chosen. Only the "Fine-Mode" and "Fine-Mode" Generic methods use a desired number of Taus as an input. For all cases where a value for the number of Taus was needed, the value of eleven was used.

After some analysis and pre-testing, it was determined that all of the methods provided a variable number of Dopplers based DFT size used for a particular method. The only method that used a direct computation for the number of Dopplers in its processing was the Filter Bank method. Therefore, no pre-determined values for the number of Dopplers were selected. The number of Dopplers was calculated only for the Filter Bank method. The number of Dopplers is determined based on the Doppler spacing. The Doppler spacing is calculated as

$$Doppler\ Spacing = \frac{4}{10T}, \tag{75}$$

where T is calculated as

$$T = \ = \frac{N}{F_s}, \tag{76}$$

where N is the number of samples (constant 11000) and F_s is the sampling rate (5.5, 11, and 22 kHz). The value of 4 in the numerator of (75) appears because this gives the null-to-null value of the main lobe of the rectangular filter/window as described in Section 6.3.1 in [9]. Each method is filtered/windowed. The rectangular filter/window is used because it provides a worse case scenario (most narrow peak) for computing the Doppler spacing. This guarantees that the Doppler spacing computed will be sufficient for all the methods. The value of 10 in the denominator of (75) appears because to compute the Doppler from the ambiguity surface, a curve fit has to be applied in the Doppler direction. The curve fit requires five points (the peak and two points on each side). This requires that there be ten points from null-to-null. Dividing 4/T by 10 guarantees that there will be enough points in the Doppler direction for the curve fit. For the purposes of testing the Doppler spacing values computed for the 5.5 kHz, 11 kHz, and 22 kHz sampling rates were 0.2, 0.4, and 0.8 respectively. After preliminary testing, the 0.8 value was not small enough because the curve fit for the 22 kHz cases were not accurate and succumbing to errors. Analysis showed that a value of 0.5 was sufficient to provide accurate curve fits for the 22 kHz cases.

The final item examined was data reduction. Data reduction techniques (e.g. decimation) allow for reducing the amount of data used in computation, but at the cost of accuracy. Decimation is incorporated into the “Fine-Mode”, “Fine-Mode” Generic, “Fine-Mode” Generic Frequency Domain”, and the 2-D Cross Spectra methods. Varying the amount of decimation in each of these techniques would give a measure as to how the performance in calculating the true Doppler and time-delay degrades. Since the *BT* =10000, a constant amount of samples is collected for each frequency given the sample rates in Table 1 (11000 samples). Therefore, three values of decimation were chosen for each bandwidth for each band (HF, VHF, and UHF) for 27 total values. The decimation applied for each sampling rate for a particular band was chosen such that the $F_s/2$ frequency after decimation would be the same. The highest decimation values for each sampling rate were limited by the fact that the decimation could not exceed the maximum filter/window length (*L*). The maximum filter/window length as stated in Chapter 3 Section 3.1 is computed as

$$\frac{F_s}{2\pi\nu_{\max}}, \tag{77}$$

where F_s is the sampling rate of the signal, and $\nu_{\max}$ is the maximum expected Doppler shift.

The low decimation rate was selected based on the fact that the “Fine-Mode” Generic Window Frequency Domain and 2-D Cross Spectra methods have a lower bound on the filter length (*L*). This lower bound is $T/2$ where T is the number of time-delays. Since L must always be $\geq$ this lower bound, it was deemed appropriate for this testing that the minimum decimation could be set to this value for all the methods. The medium decimation rate was chosen such that it would provide an $F_s/2$ frequency after decimation somewhere in the middle of the low decimation and high decimation. To get a feel for how close the $F_s/2$ frequency after decimation is to the maximum Doppler frequency, see Figure 36 which depicts the frequency representation for the narrow band HF test case. The values of decimation for each of the frequencies and sampling rates are shown in Table 1.

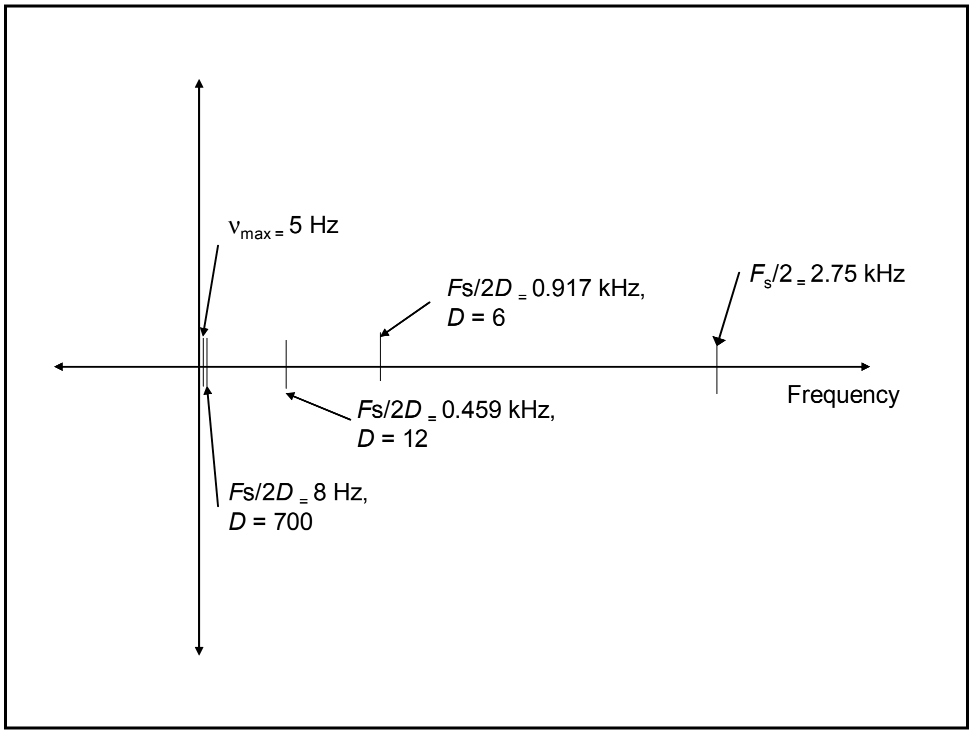
ν_{max} = 5 Hz
$F_s/2D$ = 0.917 kHz, D = 6
$F_s/2$ = 2.75 kHz
Frequency
$F_s/2D$ = 0.459 kHz, D = 12
$F_s/2D$ = 8 Hz, D = 700

Figure 36 Frequency Representation After Decimation for Narrow Band HF Test Case

Frequency (MHz)	*Bandwidth (kHz)*	*Sampling Rate (Ksps)*	*BT*	*T(sec)*	*Number of Samples*	*Desired Doppler Spacing (Hz)*	*Decimation Rate*
5	5	5.5	10000	2	11000	0.2	6 (L) 12 (M) 700 (H)
5	10	11	10000	1	11000	0.4	12 (L) 24 (M) 1400 (H)
5	20	22	10000	0.5	11000	0.5	24 (L) 48 (M) 2801 (H)
50	5	5.5	10000	2	11000	0.2	6 (L) 12 (M) 83 (H)
50	10	11	10000	1	11000	0.4	12 (L) 24 (M) 166 (H)
50	20	22	10000	0.5	11000	0.5	24 (L) 48 (M) 333 (H)
500	5	5.5	10000	2	11000	0.2	6 (L)

Frequency (MHz)	*Bandwidth (kHz)*	*Sampling Rate (Ksps)*	*BT*	*T(sec)*	*Number of Samples*	*Desired Doppler Spacing (Hz)*	*Decimation Rate*
							7 (M) 8 (H)
500	10	11	10000	1	11000	0.4	12 (L) 14 (M) 18 (H)
500	20	22	10000	0.5	11000	0.5	24 (L) 28 (M) 33 (H)

Table 1. Test Case Specifics

Thus the testing performed was in a tiered level structure as is shown in Figure 37. Note that this tiered level structure is computed for each unique frequency. Also note that the Filter Bank method does not have any decimation built into it, so from each of the bandwidths the decimation level is skipped. For testing, a tiered level structure was executed for each frequency (5 MHz, 50 MHz, and 500 MHz).

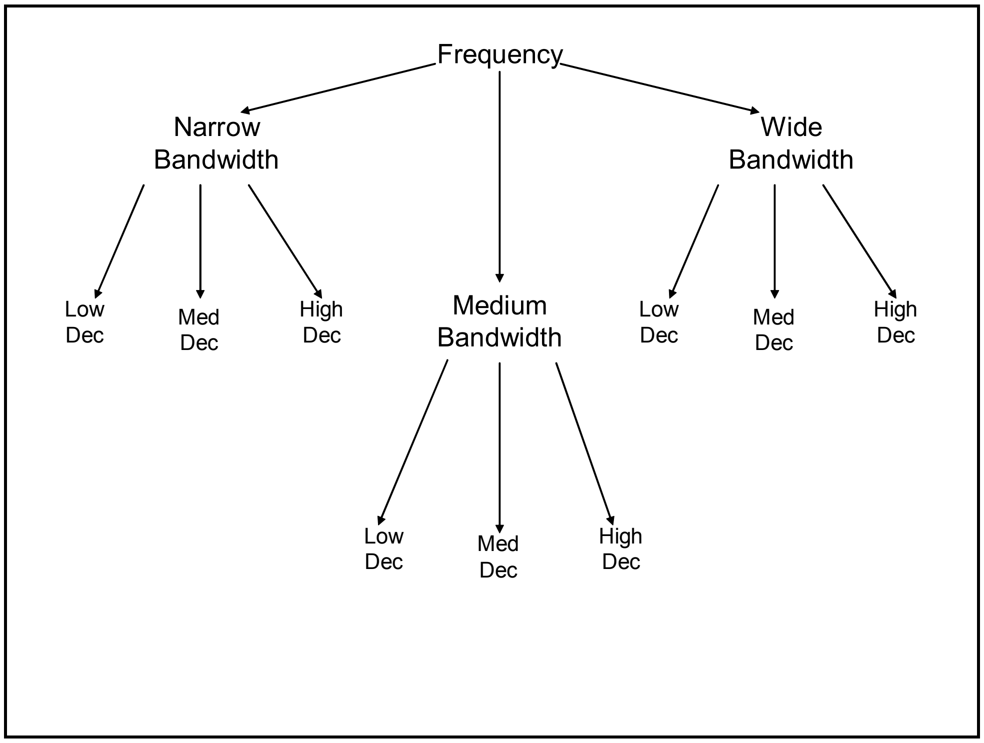

Figure 37 Testing Structure

The tool used to create and execute the simulations was Matlab Student Edition version 6.5.0.1924 Release 13. The signal processing toolbox (purchased separately from the Student Edition) was also included in the Matlab software used to execute the simulations. To create the simulations, an m-file was created that would act as a script. Flags were created in the m-file that when set with particular values, would allow for a particular case to be simulated. A pseudo-voice sequence was used to create the 9 different signals (3 signals at 3 unique frequencies and bandwidths). This sequence was complex and was first created by taking a random normally distributed noise sequence for the largest number of signal samples needed (11000) and filtering each one with a different filter (9 in total). The state of the random sequence used was set to be a constant (state 2 [arbitrarily chosen]) such that for each time the script was started, the same random sequence would be used. This was done to reduce any errors that could be introduced by using a different random sequence for each test case.

The next step taken was to design a filter for each sampling rate (9 in total). Since BT was a constant 10000, each filter designed would have the same form only a different frequency range when viewing the spectrum. This simplified the filter design to one filter. Recall that to avoid critically sampling the signal, each sampling rate used was 10% greater than the Nyquist sampling rate (F_{s_c}) or $1.1 \times F_{s_c} = F_s$. Converting from physical frequency in Hz to digital frequency n radians/sample, it is clear that the one filter will always be in the same proportion since the same cut-off frequency is used for all 9 signals, as is shown in

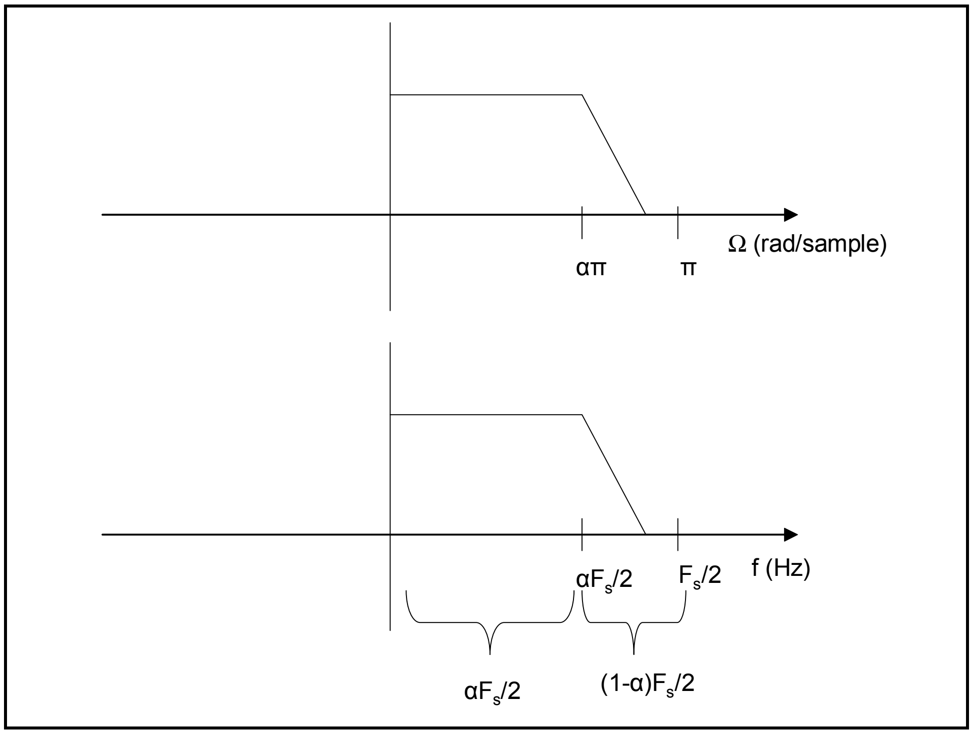

Figure 38 Filter Analysis Using Digital Frequency

Figure 38.

The Matlab fir1 command was used to design the filter. The filter had 25 taps and a cut-off frequency of 0.9091π radians/sample. The filter was applied to the pseudo-voice sequence by using the Matlab filter command, thus creating the signal s_1. Depending upon frequency and bandwidth, the s_1 signal was then frequency shifted to give the signal desired the Doppler shift. The Doppler shifted s_1 signal was the s_2 signal. A function called freq_shift was created and used to apply the Doppler shift to the s_1 signal. For simplicity of simulation, it was assumed that both signal streams produced would be at the same signal-to-noise ratio (SNR). Therefore, no manipulation of the data was performed to adjust the SNR of the second stream. Both signal streams were both upsampled by a factor of 4 to guarantee that the desired spacing in the Tau dimensions would provide for an adequate curve fit in determining the Tau measurements from the CAF peak. At this point, depending upon the flags used, different functions were executed for the different methods. The Filter Band method was coded in the fb function. The "Fine-Mode" and "Fine-Mode" Generic methods were coded in the fmg function. For simplicity of implementation, when coding the “Fine-Mode” Generic method, the corrections in the algorithm for ripple and edge effects were not implemented. The “Fine-Mode” Generic Frequency Domain method was coded in the fmgfd function, and the 2-D Cross-Spectra method was coded in the twodcs function. All the test cases, except for the Filter Bank method, for all the frequencies (HF, VHF, and UHF) were run with *L*=*D*. This was done to make the attempt to be able to compare the accuracy back to the “Fine-Mode” method, where *L* always is equal to *D*. Since all the methods, except the Filter Bank method, support *L*=*D*, all the date reduction methods could be compared on the same level. By forcing the “Fine-Mode” Generic, “Fine-Mode” Generic Frequency Domain, and 2-D Cross Spectra methods to have *L*=*D* accuracy was sacrificed to produce better computational complexity. To determine the impact of having L>D, the “Fine-Mode” Generic, “Fine-Mode” Generic Window Frequency Domain, and 2-D Cross Spectra methods were re-run with L>D for the HF Frequency. Recall that it was stated in Chapter 3 Section 3.3 that the optimal accuracy for the 2-D Cross Spectra method is obtained with $L = D/0.88$. The decision was made to run with $L > D$ such that $L > D/0.88$ in order to provide a comparison between the “Fine-Mode” Generic, “Fine-Mode” Generic Frequency Domain, and 2-D Cross Spectra methods and show how accuracy and computational complexity is affected with $L >> D$.

For the “Fine-Mode” Generic method, the filter used was designed using the Matlab fir1 command. The cut-off frequency used was the maximum expected Doppler frequency. The “Fine-Mode” Generic Frequency Domain method also was run with this same fir1 filter, in order to provide a comparison to the “Fine-Mode” Generic method, that ruled out the filter/window being used and instead focused on the method (time-domain vs. Frequency domain).

The windows used on the 2-D Cross Spectra method were chosen to be the same type of windows (Tukey). Having the same type of windows provided simplification in having to design fewer windows, since the window on the second signal stream is always a Tukey window. Using a Tukey window on the first signal stream also made sense because the Tukey window has amplitude of 1 and the less the signal stream is modified, the better. The Tukey window is similar to a rectangular window, providing a narrow peak (more desirable for curve-fitting). To be able to provide a decent comparison with the 2-D Cross

Spectra method, the "Fine-Mode" Generic Frequency Domain also was computed for all the test cases using a Tukey window on the first signal stream. This would provide the opportunity to observe the accuracy between the two methods when a second Tukey window is introduced in the 2-D Cross Spectra method.

Since the accuracy of the "Fine-Mode" Generic method is based on the filter used, the HF test cases using the "Fine-Mode" Generic method were re-run using "better" filters designed using the Matlab Filter Design tool, in order to demonstrate this. Before using the filters designed using the Matlab tool, an attempt was made to create the filters using the remez algorithm, but given that the filter lengths were much shorter than the order that remez desired for high accuracy, the remez algorithm produced poor filters. The "better" filters were designed in an ad-hoc manner, where parameters in the tool were modified until the filter looked better visually in the frequency domain than the fir1 filter (flatter stop band and more attenuation in the stop band) better results in at least one dimension (Tau or Doppler) were obtained, or results close to the fir1 filter were obtained. The purpose of this research is not to develop the "ideal" filters to use for the given test cases, but to show that accuracy is dependent upon the filter used and that by using a "better" filter should produce better accuracy. Note that some filters created could not yield better performance to the fir1 filter. This was most likely due to the stop-band attenuation not being lower than the fir1 filter or other limitations placed on these "better" filters that could not allow them to exceed or equal the performance of the fir1 filters. See Table 2 for the parameters entered into the Matlab Filter Design tool to create the "better" filters. See Table 3 in Chapter 4 Section 4.3 for a list of results from executing the test cases.

Filter Type	***Design Method***	***Filter Order***	***Density Factor***	***F_s (KHz)***	***F_{pass} (Hz)***	***F_{stop} (Hz)***	***W_{pass}***	***W_{stop}***	***Usage***
Lowpass	FIR (Equiripple)	5	50	22	5	10000	1	1	HF Narrow BW, L=D, D=6

Filter Type	*Design Method*	*Filter Order*	*Density Factor*	*F_s (KHz)*	*F_{pass} (Hz)*	*F_{stop} (Hz)*	*W_{pass}*	*W_{stop}*	*Usage*
Lowpass	FIR (Equiripple)	11	50	22	5	1500	1	0.2	HF Narrow BW L=D, D=12
Lowpass	FIR (Equiripple)	699	16	22	5	50	1	1	HF Narrow BW L=D, D=700 and L=700,D= 100
Lowpass	FIR (Equiripple)	99	16	22	5	500	1	1	HF Narrow BW L=100, D=6,12
Lowpass	FIR (Equiripple)	11	50	44	5	3000	1	0.2	HF Medium BW L=D, D=12
Lowpass	FIR (Equiripple)	23	50	44	5	3000	1	0.2	HF Medium BW L=D, D = 24
Lowpass	FIR (Equiripple)	1399	50	44	5	37	1	0.2	HF Medium BW L=D, D=1400
Lowpass	FIR (Equiripple)	199	16	44	5	1000	1	1	HF Medium BW L=200, D=12,24
Lowpass	FIR (Equiripple)	23	50	88	5	8160	1	0.2	HF Wide BW L=D, D=24
Lowpass	FIR (Equiripple)	47	50	88	5	2800	1	0.2	HF Wide BW L=D, D=48
Lowpass	FIR (Equiripple)	2800	50	88	5	98	1	0.2	HF Wide BW L=D, D=2801
Lowpass	FIR (Equiripple)	399	16	88	5	380	1	1	HF Wide BW L=400, D=24,48
Lowpass	FIR (Equiripple)	2800	16	88	5	98	1	1	HF Wide BW L=2801, D=400

Table 2. "Better" Filter Parameters for Matlab Filter Design Tool

The “Fine-Mode” Generic Frequency Domain method HF test cases were also executed with these “better” filters in order to better compare the two methods.

For the Filter Bank method, a queuing method for passing the number of Dopplers into the calculation was adopted because calculating all the Dopplers for UHF cases caused Matlab to run out of memory. The queuing method adopted made use of apriori knowledge of the expected Doppler frequency and allowed for a range of Dopplers to be entered and calculating them for this method. This queuing method applied is not unlike what may be used by actual emitter location systems.

In order to provide a metric for the accuracy produced for all the methods, the numbers of computations for each method were calculated for each test case. The number of computations were separated out into number of real adds and number of real multiplies. Real computations were calculated because processors used in emitter location systems work with real computations not complex. The equations used to compute the number of real multiplies and real adds for each complex multiply can be verified to be

$$\begin{aligned} &\#\,\mathrm{Re}al\ Multiplies = 4N \\ &\#\,\mathrm{Re}al\ Adds = 2N \end{aligned} , \tag{78}$$

where N is the number of complex multiplies. In the case where complex DFT operations were performed, a radix-2 DFT was assumed because it is the DFT via the FFT algorithm most widely used. The equations used to convert the complex DFT to real multiplies and adds are equations 5.24a and 5.24b from Section 5.3 from [2].

In order to compute the number of real computations (multiplications and adds) for the Filter Bank method, equation 2.3 from [2] which contains the number of complex multiplications was used as a starting point. This equation was converted to real multiplications and adds and thus became

$$\# \mathrm{Re}\mathit{al\ Multiplies} = (K+1)\left[4N + 4(N+T) + 4(N+T)\left[\log_2(N+T) - 2\right] + 8\right] + 2(N+T)\left[\log_2(N+T) - 2\right] + 4$$

$$\# \mathrm{Re}\mathit{al\ Adds} = (K+1)\left[2N + 2(N+T) + 6(N+T)\log_2(N+T) - 4(N+T) + 4\right] + 3(N+T)\log_2(N+T)$$,

(79)

where *K* is the number of Dopplers, *N* is the number of signal samples, and *T* is the number of time-delays.

To compute the number of real computations for the "Fine-Mode" and "Fine-Mode" Generic methods, equation 3.3 from [2] which contains the number of complex multiplications was used as base number of computations. Thus the number of real multiplications and adds can be proven to be

$$\# \mathrm{Re}\mathit{al\ Multiplies} = (T+1)\left[4N + 2(DFT\_Size)\left[\log_2(DFT\_Size) - 2\right] + 4\right]$$

$$\# \mathrm{Re}\mathit{al\ Adds} = (T+1)(2N + 4ML + 3(DFT\_Size)\log_2(DFT\_Size) - 2(DFT\_Size) + 2)$$,

(80)

for the "Fine-Mode" method and

$$\# \mathrm{Re}\mathit{al\ Multiplies} = (T+1)(4N + 4M(2L+1) + 2(DFT\_Size)\left[\log_2(DFT\_Size) - 2\right] + 4)$$

$$\# \mathrm{Re}\mathit{al\ Adds} = (T+1)(2N + 4ML + 3(DFT\_Size)\log_2(DFT\_Size) - 2(DFT\_Size) + 2)$$,

(81)

for the "Fine-Mode" Generic method, where *T* is the number of time-delays, *M* is the number of inner sums (number of blocks), *L* is the filter length, and *DFT_Size* is the size of the DFT for each inner sum. Note that there are more multiplications performed for the "Fine-Mode" Generic method. The "Fine-

Mode" method requires fewer multiplications because it makes use of the "all-ones" filter and those multiplies come for free because of the multiplication by one.

The number of computations for the "Fine-Mode" Generic Frequency Domain made use of equation 3.6 from [2] for the base number of complex computations. The number of real multiplications and adds can be proven to be

$$\begin{aligned}&\#\,\mathrm{Re}al\ Multiplies = M(4L + 6(DFT\_Size1)[\log_2(DFT\_Size1) - 2] + 12 + \\ &4(DFT\_Size1)) + DFT\_Size1[2(DFT\_Size2)[\log_2(DFT\_Size2) - 2] + 4]\end{aligned}$$

$$\begin{aligned}&\#\,\mathrm{Re}al\ Adds = M(4L + 9(DFT\_Size1)\log_2(DFT\_Size1) - 6(DFT\_Size1) + 6 + , \\ &2(DFT\_Size1)) + DFT\_Size1(3(DFT\_Size2)\log_2(DFT\_Size2) - \\ &2(DFT\_Size2) + 2)\end{aligned} \tag{82}$$

where M is the number of blocks, L is the window/filter length, *DFT_Size1* is the size of the DFT of both signal streams (f_m and s_2) and *DFT_Size2* is the size of the IDFT of the multiplication of F_m and S_2^* (see Figure 31).

The 2-D Cross Spectra method also made use of equation 3.6 from [2] for its original base number of complex computations. This equation was slightly modified to account for the 2nd stream being windowed. The number of real multiplications and adds are

$$\# \mathrm{Re}al\ Multiplies = M(8L_{w_2} + 4(DFT\_Size)[\log_2(DFT\_Size) - 2] + 8 + \\ 4(DFT\_Size)) + DFT\_Size(2(DFT\_Size\_Dopp)[\log_2(DFT\_Size\_Dopp) - 2] \\ + 4) + DFT\_Size\_Dopp(2(DFT\_Size\_Tau)[\log_2(DFT\_Size\_Dopp) - 2] + 4)$$

$$\# \mathrm{Re}al\ Adds = M(4L_{w_2} + 6(DFT\_Size)\log_2(DFT\_Size) - 2(DFT\_Size) + 4 + \\ 2(DFT\_Size)) + DFT\_Size(3(DFT\_Size\_Dopp)\log_2(DFT\_Size\_Dopp) - \\ 2(DFT\_Size\_Dopp) + 2) + \\ DFT\_Size\_Dopp(3(DFT\_Size\_Tau)\log_2(DFT\_Size\_Tau) - \\ 2(DFT\_Size\_Tau) + 2) \quad ,$$

(83)

where M is the number of blocks, L_{w_2} is the window of for s_2, *DFT_Size* is the size of the DFT of both windowed signal streams (s_1 and s_2), *DFT_Size_Dopp* is the size of the DFT of the columns of the *CS* matrix (see Figure 33), and *DFT_Size_Tau* is the size of the IDFT of the rows of the A_{Doppler} matrix (see Figure 33).

In order to compute the time-delay and Doppler for all the test cases, two curve-fits were applied to the ambiguity function computed, one in the time-delay direction and the other in the Doppler direction. To apply the curve-fits, five points around the peak (the peak and two points on either side of the peak) were selected and then put through the Matlab polyfit command to fit a quadratic to the data. During preliminary testing, the polyfit was producing poorly conditioned output polynomials. Therefore, the curve-fit was modified to use the polyfit command with command option of centering and scaling the curve-fit on the data. According to the Matlab help files provides better precision to the curve-fit. Since the data was centered on the true Doppler, the Doppler error calculation simplified from

$$Doppler\_Error = \sqrt{(Calculated\_Doppler - True\_Doppler\_Frequency)^2} \ ,$$

(84)

to

$$Doppler\_Error = \sqrt{(Calculated\_Dopplery)^2}\ , \qquad (85)$$

The Matlab code for the all the functions as well as the main script, without the “better filter” logic which was kludged to produce results, can be viewed in the Appendices.

4.3 Summary of Results

The results of all the tests executed are summarized in Table 3. The time-delay error and Doppler error columns have been color-coded to rate the results. The color code is green is good, yellow is degraded, and red is bad. The color codes are meant to be used on a column-by-column basis and should not be read across the rows. The color codes are applied as follows: any time-delay error < 1 us or Doppler error < 1 Hz is deemed good. Any time-delay error ≥ 1 us but < 20 us or Doppler error ≥ 1 Hz but < 20 Hz is deemed degraded. Any time-delay error ≥ 20 us or Doppler error ≥ 20 Hz is deemed bad. The ideal level of error for the time-delay and Doppler is 0 us and 0 Hz, respectively. However because of errors introduced by decimation, the desired values of error and time-delay were < 1 us and < 1 Hz respectively.

Frequency (MHz)	***Method***	***F_s (Hz)***	***D***	***L***	***τ Error (µs)***	***Doppler Error (Hz)***	***# Real Adds***	***# Real Multiplies***
5	FB	22000	N/A	N/A	0	0	697,861,722	452,686,903
5	FB	44000	N/A	N/A	0	0	365,462,807	236,954,093
5	FB	88000	N/A	N/A	0	0	298,983,025	193,807,531
5	FM	22000	6	6	0.203434	0.113341	20,076,504	12,335,664
			12	12	0.403557	0.113305	11,032,536	6,830,640
			700	700	73.750255	0.264644	3,240,408	2,148,912
5	FM	44000	12	12	0.438876	0.245003	11,032,536	6,830,640
			24	24	0.902120	0.245004	6,806,040	4,274,736
			1400	1400	279.987817	0.125448	3,235,608	2,127,408
5	FM	88000	24	24	0.314139	0.178531	6,806,040	4,274,736
			48	48	0.620008	0.178579	4,839,960	3,095,088
			2801	2801	333.138259	0.061644	3,219,480	2,118,192
5	FMG[1]	22000	6	6	0.028005	0.113360	20,252,520	16,912,080
			12	12	12.655625	0.113340	11,120,544	11,231,040
			700	700	52.517265	0.262709	3,241,920	6,385,536

[1] Filter used for test case was designed using Matlab fir1 command

Frequency (MHz)	***Method***	***F_s (Hz)***	***D***	***L***	***τ Error (μs)***	***Doppler Error (Hz)***	***# Real Adds***	***# Real Multiplies***
5	FMG[1]	44000	12	12	5.969105	0.245018	11,120,544	11,231,040
			24	24	1.468118	0.244926	6,850,056	8,588,304
			1400	1400	201.766702	0.125538	3,236,376	6,429,744
5	FMG[1]	88000	24	24	0.181176	0.178517	6,850,056	8,588,304
			48	48	0.910654	0.178509	4,861,968	7,364,640
			2801	2801	171.796572	0.061644	3,219,864	6,421,296
5	FMG[1]	22000	6	100	0.188807	0.113383	53,266,344	82,939,728
			12	100	0.290949	0.113380	26,576,184	42,142,320
			100	700	1.463614	0.045306	16,410,696	31,740,048
5	FMG[1]	44000	12	200	0.433369	0.244906	44,057,568	77,105,088
			24	200	0.859447	0.244909	22,266,696	39,421,584
			200	1400	0.500552	0.025317	15,786,024	31,080,528
5	FMG[1]	88000	24	400	0.301804	0.178684	39,642,504	74,173,200
			48	400	0.613012	0.178686	20,221,032	38,082,768
			400	2801	0.212516	0.018019	15,194,712	30,168,240
5	FMG[2]	22000	6	6	0.040020	0.113359	20,252,520	16,912,080
			12	12	0.446975	0.113305	11,120,544	11,231,040
			700	700	11.643097	0.263071	3,241,920	6,385,536
5	FMG[2]	44000	12	12	0.434742	0.244995	11,120,544	11,231,040
			24	24	0.706200	0.244941	6,850,056	8,588,304
			1400	1400	1.450071	0.125611	3,236,376	6,429,744
5	FMG[2]	88000	24	24	0.067634	0.178515	6,850,056	8,588,304
			48	48	0.523144	0.178527	4,861,968	7,364,640
			2801	2801	2.527392	0.065953	3,219,864	6,421,296
5	FMG[2]	22000	6	100	0.199295	0.113379	53,266,344	82,939,728
			12	100	0.371750	0.113378	26,576,184	42,142,320
			100	700	0.901442	0.045704	16,410,696	31,740,048
5	FMG[2]	44000	12	200	0.436841	0.244948	44,057,568	77,105,088
			24	200	0.873686	0.244948	22,266,696	39,421,584
			200	1400	1.273246	0.022203	15,786,024	31,080,528
5	FMG[2]	88000	24	400	0.256963	0.178709	39,642,504	74,173,200
			48	400	0.503396	0.178697	20,221,032	38,082,768
			400	2801	0.293661	0.012058	15,194,712	30,168,240
5	FMGFD[3]	22000	6	6	1.961396	0.113341	26,431,436	15,773,080
			12	12	12.587743	0.113326	25,892,698	15,384,628
			700	700	28.659983	0.262602	23,884,290	13,960,196
5	FMGFD[3]	44000	12	12	5.772418	0.245024	25,892,698	15,384,628
			24	24	0.421330	0.244913	25,366,620	15,019,192
			1400	1400	193.979748	0.125223	23,690,944	13,827,456
5	FMGFD[3]	88000	24	24	1.259093	0.178523	25,366,620	15,019,192
			48	48	0.852090	0.178508	24,844,766	14,664,124
			2801	2801	174.374113	0.061878	23,305,856	13,581,568
5	FMGFD[3]	22000	6	100	0.203715	0.113371	489,609,908	296,056,168
			12	100	0.398849	0.113367	232,239,912	139,650,640
			100	700	0.504922	0.043591	215,076,764	128,278,328
5	FMGFD[3]	44000	12	200	0.428097	0.245093	480,964,674	290,281,604
			24	200	0.869259	0.245084	227,919,852	136,765,400
			200	1400	0.081991	0.021598	209,209,764	124,425,032
5	FMGFD[3]	88000	24	400	0.208732	0.178702	471,946,204	284,288,952
			48	400	0.435327	0.178676	223,480,084	133,812,776
			400	2801	0.980649	0.010554	201,729,600	119,557,248
5	FMGFD[4]	22000	6	6	1.877667	0.113341	26,431,436	15,773,080
			12	12	10.892650	0.113327	25,892,698	15,384,628
			700	700	53.583883	0.262675	23,884,290	13,960,196
5	FMGFD[4]	44000	12	12	4.937427	0.245023	25,892,698	15,384,628
			24	24	1.635516	0.244922	25,366,620	15,019,192

[2] Filter used for test case was designed using Matlab Filter Design Tool
[3] Window used for test case was a Tukey window
[4] Same fir1 filter used as for FMG

Frequency (MHz)	***Method***	***F_s (Hz)***	***D***	***L***	***τ Error (μs)***	***Dop-pler Error (Hz)***	***# Real Adds***	***# Real Multiplies***
			1400	1400	200.623807	0.125516	23,690,944	13,827,456
5	FMGFD[4]	88000	24	24	0.194307	0.178514	25,366,620	15,019,192
			48	48	0.872296	0.178512	24,844,766	14,664,124
			2801	2801	171.373295	0.061655	23,305,856	13,581,568
5	FMGFD[4]	22000	6	100	0.202034	0.113372	489,609,908	296,056,168
			12	100	0.408525	0.113367	232,239,912	139,650,640
			100	700	0.729742	0.043614	215,076,764	128,278,328
5	FMGFD[4]	44000	12	200	0.429230	0.245094	480,964,674	290,281,604
			24	200	0.877294	0.245086	227,919,852	136,765,400
			200	1400	0.126728	0.021616	209,209,764	124,425,032
5	FMGFD[4]	88000	24	400	0.209519	0.178701	471,946,204	284,288,952
			48	400	0.445662	0.178679	223,480,084	133,812,776
			400	2801	1.103619	0.010625	201,729,600	119,557,248
5	FMGFD[5]	22000	6	6	1.670361	0.113342	26,431,436	15,773,080
			12	12	0.522479	0.113301	25,892,698	15,384,628
			700	700	6.320595	0.262803	23,884,290	13,960,196
5	FMGFD[5]	44000	12	12	0.624674	0.245005	25,892,698	15,384,628
			24	24	0.828785	0.244940	25,366,620	15,019,192
			1400	1400	341.533573	0.126409	23,690,944	13,827,456
5	FMGFD[5]	88000	24	24	0.117422	0.178512	25,366,620	15,019,192
			48	48	0.577472	0.178529	24,844,766	14,664,124
			2801	2801	11.362605	0.065900	23,305,856	13,581,568
5	FMGFD[5]	22000	6	100	0.198511	0.113371	489,609,908	296,056,168
			12	100	0.419154	0.113367	232,239,912	139,650,640
			100	700	3.462016	0.043476	215,076,764	128,278,328
5	FMGFD[5]	44000	12	200	0.441033	0.245082	480,964,674	290,281,604
			24	200	0.893065	0.245073	227,919,852	136,765,400
			200	1400	0.335380	0.021781	209,209,764	124,425,032
5	FMGFD[5]	88000	24	400	0.213101	0.178698	471,946,204	284,288,952
			48	400	0.386859	0.178674	223,480,084	133,812,776
			400	2801	0.762588	0.010871	201,729,600	119,557,248
5	2DCS	22000	6	6	471.928694	0.119615	141,304,060	82,049,784
			12	12	536.071746]	0.114210	67,624,872	39,163,216
			700	700	271.531861	0.262251	36,531,484	21,346,872
5	2DCS	44000	12	12	267.974226	0.244162	67,624,872	39163216
			24	24	233.660125	0.244917	68,913,712	398,63,904
			1400	1400	348.710958	0.126381	37,458,688	21,964,288
5	2DCS	88000	24	24	116.021828	0.178514	68,913,712	39,863,904
			48	48	95.916826	0.178476	70,268,488	40,687,760
			2801	2801	186.022185	0.061972	38,252,352	22,503,040
5	2DCS	22000	6	100	19.820280	0.113386	1,347,144,896	799,152,512
			12	100	39.655814	0.113382	648,435,340	382,817,560
			100	700	38.572067	0.043617	320,122,532	189,562,184
5	2DCS	44000	12	200	9.521413	0.245123	1,368,745,920	8,132,647,68
			24	200	19.060423	0.245110	659,270,656	389,896,192
			200	1400	19.085851	0.021597	324,376,272	192,408,992
5	2DCS	88000	24	400	4.075161	0.178760	1,389,658,280	827,105,616
			48	400	8.137236	0.178749	669,733,976	396,822,704
			400	2801	9.867400	0.010686	327,326,964	194,427,368
50	FB	22000	N/A	N/A	0	0	1,362,659,551	884,152,522
50	FB	44000	N/A	N/A	0	0	697,861,722	452,686,903
50	FB	88000	N/A	N/A	0	0	564,902,156	366,393,779
50	FM	22000	6	6	0.013189	0.007319	20,076,504	12,335,664
			12	12	0.000647	0.007128	11,032,536	6,830,640
			83	83	1.806011	0.220343	3,933,384	2,554,416
50	FM	44000	12	12	0.546796	0.301155	11,032,536	6,830,640
			24	24	1.232829	0.301164	6,806,040	4,274,736

[5] Filter designed using Matlab Filter Design Tool, same as for FMG

Frequency (MHz)	***Method***	***F_s (Hz)***	***D***	***L***	***τ Error (μs)***	***Doppler Error (Hz)***	***# Real Adds***	***# Real Multiplies***
			166	166	14.029722	0.108862	3,519,576	2,308,656
50	FM	88000	24	24	0.242958	0.151487	6,806,040	4,274,736
			48	48	0.444009	0.151745	4,839,960	3,095,088
			333	333	3.936305	0.001778	3,525,960	2,308,656
50	FMG[1]	22000	6	6	0.166738	0.007340	20,252,520	16,912,080
			12	12	12.430606	0.007234	11,120,544	11,231,040
			83	83	8.083229	0.221603	3,946,128	6,810,912
50	FMG[1]	44000	12	12	5.907726	0.301125	11,120,544	11,231,040
			24	24	1.772533	0.300911	6,850,056	8,588,304
			166	166	7.230216	0.109366	3,525,960	6,560,400
50	FMG[1]	88000	24	24	0.109891	0.151560	6,850,056	8,588,304
			48	48	0.720542	0.151645	4,861,968	7,364,640
			333	333	5.585045	0.000779	3,529,152	6,566,784
50	FMGFD[3]	22000	6	6	1.896360	0.007329	26,431,436	15,773,080
			12	12	12.262696	0.007161	25,892,698	153,84,628
			83	83	7.169745	0.221698	25,588,420	15,058,312
50	FMGFD[3]	44000	12	12	5.685575	0.301164	25,892,698	15,384,628
			24	24	0.127705	0.300841	25,366,620	15,019,192
			166	166	8.201645	0.109351	25,257,780	14,835,304
50	FMGFD[3]	88000	24	24	1.190068	0.151592	25,366,620	15,019,192
			48	48	0.612315	0.151666	24,844,766	14,664,124
			333	333	7.218809	0.000776	41,164,064	24,042,048
50	FMGFD[4]	22000	6	6	1.809923	0.007331	26,431,436	15,773,080
			12	12	10.579849	0.007169	25,892,698	15,384,628
			83	83	7.854864	0.221589	25,588,420	15,058,312
50	FMGFD[4]	44000	12	12	4.853693	0.301159	25,892,698	15,384,628
			24	24	1.920431	0.300903	25,366,620	15,019,192
			166	166	7.291491	0.109390	25,257,780	14,835,304
50	FMGFD[4]	88000	24	24	0.126921	0.151559	25,366,620	15,019,192
			48	48	0.653187	0.151652	24,844,766	14,664,124
			333	333	5.569979	0.000857	41,164,064	24,042,048
50	2DCS	22000	6	6	472.101980	0.013532	141,304,060	82,049,784
			12	12	536.056858	0.008140	67,624,872	39,163,216
			83	83	349.286349	0.221732	34,554,672	20,037,216
50	2DCS	44000	12	12	267.971858	0.300257	67,624,872	39,163,216
			24	24	233.948641	0.300885	68,913,712	39,863,904
			166	166	158.616845	0.109118	35,351,316	20,545,064
50	2DCS	88000	24	24	116.091211	0.151565	68,913,712	39,863,904
			48	48	96.019831	0.151576	70,268,488	40,687,760
			333	333	64.265958	0.000459	67,061,472	38,882,752
500	FB	22000	N/A	N/A	$1x10^{-6}$	$5x10^{-6}$	1,362,659,551	884,152,522
500	FB	44000	N/A	N/A	0	$2x10^{-6}$	697,861,722	452,686,903
500	FB	88000	N/A	N/A	0	$3x10^{-6}$	564,902,156	366,393,779
500	FM	22000	6	6	0.419396	0.224765	20,076,504	12,335,664
			7	7	0.635383	0.209007	11,032,440	6,830,640
			8	8	0.150217	0.240312	11,032,344	6,830,640
500	FM	44000	12	12	0.052927	0.109282	11,032,536	6,830,640
			14	14	0.822300	0.163491	6,805,368	4,274,736
			18	18	1.157910	0.064482	6,805,752	4,274,736
500	FM	88000	24	24	0.983472	0.247195	6,806,040	4,274,736
			28	28	0.355426	0.183001	6,806,040	4,274,736
			33	33	0.958216	0.112369	6,806,328	4,274,736
500	FMG[1]	22000	6	6	0.671609	0.224621	20,252,520	16,912,080
			7	7	3.563571	0.208471	11,183,304	11,356,560
			8	8	2.172152	0.239282	11,164,344	11,318,640
500	FMG[1]	44000	12	12	6.235831	0.110565	11,120,544	11,231,040
			14	14	2.857153	0.163854	6,880,800	8,649,792
			18	18	1.586039	0.063606	6,864,432	8,617,056
500	FMG[1]	88000	24	24	0.681083	0.245018	6,850,056	8,588,304
			28	28	0.957560	0.185293	6,843,768	8,575,728

Frequency (MHz)	*Method*	*F_s (Hz)*	*D*	*L*	*τ Error (μs)*	*Doppler Error (Hz)*	*# Real Adds*	*# Real Multiplies*
			33	33	1.142923	0.111754	6,838,344	8,564,880
500	FMGFD[3]	22000	6	6	1.313849	0.224241	26,431,436	15,773,080
			7	7	2.165415	0.208630	13,829,944	8,152,176
			8	8	1.258024	0.240476	13,422,792	7,941,520
500	FMGFD[3]	44000	12	12	5.895590	0.109808	25,892,698	15,384,628
			14	14	2.098840	0.163774	13,929,870	8,193,692
			18	18	0.175996	0.062676	27,325,474	16,121,412
500	FMGFD[3]	88000	24	24	1.500222	0.244327	25,366,620	15,019,192
			28	28	1.033181	0.185804	24,526,648	14,546,544
			33	33	1.258652	0.111416	48,967,984	29,066,848
500	FMGFD[4]	22000	6	6	1.245058	0.224350	26,431,436	15,773,080
			7	7	2.065287	0.208663	13,829,944	8,152,176
			8	8	0.403180	0.240375	13,422,792	7,941,520
500	FMGFD[4]	44000	12	12	5.189911	0.109908	25,892,698	15,384,628
			14	14	2.222264	0.163833	13,929,870	8,193,692
			18	18	0.853834	0.062767	27,325,474	16,121,412
500	FMGFD[4]	88000	24	24	0.450447	0.244974	25,366,620	15,019,192
			28	28	1.059172	0.185607	24,526,648	14,546,544
			33	33	0.670270	0.111178	48,967,984	29,066,848
500	2DCS	22000	6	6	477.922349	0.218929	141,304,060	82,049,784
			7	7	522.907829	0.206698	73,809,616	42,820,000
			8	8	543.668976	0.243241	71,950,752	41,714,752
500	2DCS	44000	12	12	268.733374	0.109960	67,624,872	39,163,216
			14	14	269.279843	0.167318	35,432,552	20,562,640
			18	18	266.148526	0.065335	72,227,776	41,799,552
500	2DCS	88000	24	24	115.251124	0.244812	68,913,712	39,863,904
			28	28	112.199927	0.185861	67,490,508	390,29,656
			33	33	106.784758	0.112195	66,196,364	38,269,208

Table 3. Test Results

Notes:
FB = Filter Bank
FM = Fine Mode
FMG = Fine Mode Generic
FMGFD = Fine Mode Generic Frequency Domain
2DCS = 2-D Cross Spectra
[1] Filter used for test case was designed using Matlab fir1 command
[2] Filter used for test case was designed using Matlab Filter Design Tool
[3] Window used for test case was a Tukey window
[4] Filter used for test case was same one as for FMG
[5] Filter designed using Matlab Filter Design Tool, same as for FMG
🟩 Good accuracy

🟨 Degraded accuracy

🟥 Bad accuracy

All of the methods with the exception of the Filter Bank method were executed for three decimation rates (low, medium, and high), for each bandwidth (narrow, medium, and wide) for each frequency (HF, VHF, and UHF). There were a total of 216 tests. When executing the "Fine-Mode", "Fine-Mode"

Generic, and "Fine-Mode" Generic Frequency Domain methods the filter/window lengths (*L*) were set equal to decimation rate (*D*). For the 2-D Cross Spectra method the window length for s_1 was set equal to the decimation rate. The purpose of doing this was to provide a common link between all the methods (same decimation). Though this was inefficient in terms of accuracy for some of the methods ("Fine-Mode" Generic, "Fine-Mode" Generic Frequency Domain, and 2-D Cross Spectra have their worst accuracy when $L = D$), it was made up in terms of computational complexity.

The "Fine-Mode" Generic, "Fine-Mode" Generic Frequency Domain, and 2-D Cross Spectra methods were all re-run for the HF frequency test cases with decimation rates such that $L > D$ and better filters for the "Fine-Mode" Generic and "Fine-Mode" Generic Frequency Domain methods. This was executed in order to observe the effects of having a better filter and thus trying to observe better accuracy (trading computational complexity vs. accuracy). As mentioned in Chapter 3 Section 3.3, the optimum value of *L* for the 2-D Cross Spectra method is $L = D/0.88$ was chosen because this reduces the aliasing caused by decimation and windowing (allows for all the nulls from the spectral replicas to line up at one point when the true time-delay (τ) $\neq 0$). The value of *L* for the 2-D Cross Spectra method was run at L=D and then at $L >D$, so that it could be better compared to the "Fine-Mode" Generic and "Fine-Mode" Generic Frequency Domain methods (common link between all the methods was the decimation and filter/window lengths). Doing this for the 2-D Cross Spectra test set was intuitive because since there was no time-delay imparted to the signal data, all the nulls from the decimation and windowing would line up. As can be seen in Table 3, for the 2-D Cross Spectra method, when $L > D$ accuracy did improve, but not to the desired amount (< 1 us of error).

As can be seen from Table 3, the Filter Bank method produced the best accuracy among all the methods at the cost that it was the most computationally complex overall when L=D for all the other methods. This makes sense because the Filter Bank method is a "brute-force" method that does not have any decimation. Without decimation, very good accuracy is expected.

Overall, when L=D, the "Fine-Mode" method was the least computationally complex followed closely by the "Fine-Mode" Generic. The "Fine-Mode" Generic Frequency Domain and then the 2-D

Cross Spectra methods were the next best computationally complex methods, respectively, with the 2-D Cross Spectra method being the 2nd worst to the Filter Bank Method in terms of computational complexity. When viewing all the methods for L > D, the 2-D Cross Spectra method became the most computationally complex with the "Fine-Mode" Generic Frequency Domain and Filter Bank methods being the next closest (toss-up between the two) with the "Fine-Mode" Generic and "Fine-Mode" being the 2nd and 1st least computationally complex methods. For all cases (*L=D* and *L > D*), the "Fine-Mode" method was the least computationally complex.

After the Filter Bank method, in terms of accuracy, depending upon the test case, the results varied as to which method was best. Table 4 through Table 30 are the results of extracted the data from Table 3 and examining them on a case-by-case basis to determine the best method overall. For each test case, the best method for TDOA and FDOA were selected when *L=D*. The selection was based on the lowest errors with some small range. These methods have their errors highlighted in yellow. For the HF test cases, those test cases for *L>D* were also examined and the best methods when considering both *L>D* and *L=D* where selected. These methods have their errors highlighted in light green. In the special case that the *L=D* error or the *L=D* and *L>D* examinations yielded the same number, the errors are highlighted in teal. The notes for Table 4 through Table 30 as well as the color codes are listed after Table 30.

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (μs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
HF	5,000	FM	6	6	0.203434	0.113341	20,076,504	12,335,664
HF	5,000	FMG[1]	6	6	0.028005	0.113360	20,252,520	16,912,080
HF	5,000	FMG[1]	100	6	0.188807	0.113383	53,266,344	82,939,728
HF	5,000	FMG[2]	6	6	0.040020	0.113359	20,252,520	16,912,080
HF	5,000	FMG[2]	100	6	0.199295	0.113379	53,266,344	82,939,728
HF	5,000	FMGFD[4]	6	6	1.877667	0.113341	26,431,436	15,773,080
HF	5,000	FMGFD[4]	100	6	0.202034	0.113372	489,609,908	296,056,168
HF	5,000	FMGFD[5]	6	6	1.670361	0.113342	26,431,436	15,773,080
HF	5,000	FMGFD[5]	100	6	0.198511	0.113371	489,609,908	296,056,168
HF	5,000	FMGFD[3]	6	6	1.961396	0.113341	26,431,436	15,773,080
HF	5,000	FMGFD[3]	100	6	0.203715	0.113371	489,609,908	296,056,168
HF	5,000	2DCS	6	6	471.928694	0.119615	141,304,060	82,049,784

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
HF	5,000	2DCS	100	6	19.82028	0.113386	1,347,144,896	799,152,512

Table 4 HF Narrow Band Low Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA Error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
HF	5,000	FM	12	12	0.403557	0.113305	11,032,536	12
HF	5,000	FMG[1]	12	12	12.655625	0.113340	11,120,544	12
HF	5,000	FMG[1]	100	12	0.290949	0.113380	26,576,184	100
HF	5,000	FMG[2]	12	12	0.446975	0.113305	11,120,544	12
HF	5,000	FMG[2]	100	12	0.371750	0.113378	26,576,184	100
HF	5,000	FMGFD[4]	12	12	10.892650	0.113327	25,892,698	12
HF	5,000	FMGFD[4]	100	12	0.408525	0.113367	232,239,912	100
HF	5,000	FMGFD[5]	12	12	0.522479	0.113301	25,892,698	12
HF	5,000	FMGFD[5]	100	12	0.419154	0.113367	232,239,912	100
HF	5,000	FMGFD[3]	12	12	0.398849	0.113367	232,239,912	12
HF	5,000	FMGFD[3]	100	12	12.587743	0.113326	25,892,698	100
HF	5,000	2DCS	12	12	536.071746	0.114210	67,624,872	12
HF	5,000	2DCS	100	12	39.655814	0.113382	648,435,340	100

Table 5 HF Narrow Band Medium Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA Error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
HF	5,000	FM	700	700	73.750255	0.264644	3,240,408	2,148,912
HF	5,000	FMG[1]	700	700	52.517265	0.262709	3,241,920	6,385,536
HF	5,000	FMG[1]	700	100	1.463614	0.045306	16,410,696	31,740,048
HF	5,000	FMG[2]	700	700	11.643097	0.263071	3,241,920	6,385,536
HF	5,000	FMG[2]	700	100	0.901442	0.045704	16,410,696	31,740,048
HF	5,000	FMGFD[4]	700	700	53.583883	0.262675	23,884,290	13,960,196
HF	5,000	FMGFD[4]	700	100	0.729742	0.043614	215,076,764	128,278,328
HF	5,000	FMGFD[5]	700	700	6.320595	0.262803	23,884,290	13,960,196
HF	5,000	FMGFD[5]	700	100	3.462016	0.043476	215,076,764	128,278,328
HF	5,000	FMGFD[3]	700	700	28.659983	0.262602	23,884,290	13,960,196
HF	5,000	FMGFD[3]	700	100	0.504922	0.043591	215,076,764	128,278,328
HF	5,000	2DCS	700	700	271.531861	0.262251	36,531,484	21,346,872
HF	5,000	2DCS	700	100	38.572067	0.043617	320,122,532	189,562,184

Table 6 HF Narrow Band High Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA Error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
HF	10,000	FM	12	12	0.438876	0.245003	11,032,536	6,830,640
HF	10,000	FMG[1]	12	12	5.969105	0.245018	11,120,544	11,231,040
HF	10,000	FMG[1]	200	12	0.433369	0.244906	44,057,568	77,105,088
HF	10,000	FMG[2]	12	12	0.434742	0.244995	11,120,544	11,231,040
HF	10,000	FMG[2]	200	12	0.436841	0.244948	44,057,568	77,105,088

HF	10,000	FMGFD^4	12	12	4.937427	0.245023	25,892,698	15,384,628
HF	10,000	FMGFD^4	200	12	0.429230	0.245094	480,964,674	290,281,604
HF	10,000	FMGFD^5	12	12	0.624674	0.245005	25,892,698	15,384,628
HF	10,000	FMGFD^5	200	12	0.441033	0.245082	480,964,674	290,281,604
HF	10,000	FMGFD^3	12	12	5.772418	0.245024	25,892,698	15,384,628
HF	10,000	FMGFD^3	200	12	0.428097	0.245093	480,964,674	290,281,604
HF	10,000	2DCS	12	12	267.974226	0.244162	67,624,872	39163216
HF	10,000	2DCS	200	12	9.521413	0.245123	1,368,745,920	8,132,647,68

Table 7 HF Medium Band Low Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA Error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Mul-tiplies*
HF	10,000	FM	24	24	0.902120	0.245004	6,806,040	4,274,736
HF	10,000	FMG^1	24	24	1.468118	0.244926	6,850,056	8,588,304
HF	10,000	FMG^1	200	24	0.859447	0.244909	22,266,696	39,421,584
HF	10,000	FMG^2	24	24	0.706200	0.244941	6,850,056	8,588,304
HF	10,000	FMG^2	200	24	0.873686	0.244948	22,266,696	39,421,584
HF	10,000	FMGFD^4	24	24	1.635516	0.244922	25,366,620	15,019,192
HF	10,000	FMGFD^4	200	24	0.877294	0.245086	227,919,852	136,765,400
HF	10,000	FMGFD^5	24	24	0.828785	0.244940	25,366,620	15,019,192
HF	10,000	FMGFD^5	200	24	0.893065	0.245073	227,919,852	136,765,400
HF	10,000	FMGFD^3	24	24	0.421330	0.244913	25,366,620	15,019,192
HF	10,000	FMGFD^3	200	24	0.869259	0.245084	227,919,852	136,765,400
HF	10,000	2DCS	24	24	233.660125	0.244917	68,913,712	398,63,904
HF	10,000	2DCS	200	24	19.060423	0.245110	659,270,656	389,896,192

Table 8 HF Medium Band Medium Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA Error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Mul-tiplies*
HF	10,000	FM	1400	1400	279.987817	0.125448	3,235,608	2,127,408
HF	10,000	FMG^1	1400	1400	201.766702	0.125538	3,236,376	6,429,744
HF	10,000	FMG^1	1400	200	0.500552	0.025317	15,786,024	31,080,528
HF	10,000	FMG^2	1400	1400	1.450071	0.125611	3,236,376	6,429,744
HF	10,000	FMG^2	1400	200	1.273246	0.022203	15,786,024	31,080,528
HF	10,000	FMGFD^4	1400	1400	200.623807	0.125516	23,690,944	13,827,456
HF	10,000	FMGFD^4	1400	200	0.126728	0.021616	209,209,764	124,425,032
HF	10,000	FMGFD^5	1400	1400	341.533573	0.126409	23,690,944	13,827,456
HF	10,000	FMGFD^5	1400	200	0.335380	0.021781	209,209,764	124,425,032
HF	10,000	FMGFD^3	1400	1400	193.979748	0.125223	23,690,944	13,827,456
HF	10,000	FMGFD^3	1400	200	0.081991	0.021598	209,209,764	124,425,032
HF	10,000	2DCS	1400	1400	348.710958	0.126381	37,458,688	21,964,288
HF	10,000	2DCS	1400	200	19.085851	0.021597	324,376,272	192,408,992

Table 9 HF Medium Band High Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA Error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Mul-tiplies*
HF	20,000	FM	24	24	0.314139	0.178531	6,806,040	4,274,736

HF	20,000	FMG[1]	24	24	0.181176	0.178517	6,850,056	8,588,304
HF	20,000	FMG[1]	400	24	0.301804	0.178684	39,642,504	74,173,200
HF	20,000	FMG[2]	24	24	0.067634	0.178515	6,850,056	8,588,304
HF	20,000	FMG[2]	400	24	0.256963	0.178709	39,642,504	74,173,200
HF	20,000	FMGFD[4]	24	24	0.194307	0.178514	25,366,620	15,019,192
HF	20,000	FMGFD[4]	400	24	0.209519	0.178701	471,946,204	284,288,952
HF	20,000	FMGFD[5]	24	24	0.117422	0.178512	25,366,620	15,019,192
HF	20,000	FMGFD[5]	400	24	0.213101	0.178698	471,946,204	284,288,952
HF	20,000	FMGFD[3]	24	24	1.259093	0.178523	25,366,620	15,019,192
HF	20,000	FMGFD[3]	400	24	0.208732	0.178702	471,946,204	284,288,952
HF	20,000	2DCS	24	24	116.021828	0.178514	68,913,712	39,863,904
HF	20,000	2DCS	400	24	4.075161	0.178760	1,389,658,280	827,105,616

Table 10 HF Wide Band Low Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA Error (µs)***	***FDOA error (Hz)***	***Real Adds***	***Real Mul-tiplies***
HF	20,000	FM	48	48	0.620008	0.178579	4,839,960	3,095,088
HF	20,000	FMG[1]	48	48	0.910654	0.178509	4,861,968	7,364,640
HF	20,000	FMG[1]	400	48	0.613012	0.178686	20,221,032	38,082,768
HF	20,000	FMG[2]	48	48	0.523144	0.178527	4,861,968	7,364,640
HF	20,000	FMG[2]	400	48	0.503396	0.178697	20,221,032	38,082,768
HF	20,000	FMGFD[4]	48	48	0.872296	0.178512	24,844,766	14,664,124
HF	20,000	FMGFD[4]	400	48	0.445662	0.178679	223,480,084	133,812,776
HF	20,000	FMGFD[5]	48	48	0.577472	0.178529	24,844,766	14,664,124
HF	20,000	FMGFD[5]	400	48	0.386859	0.178674	223,480,084	133,812,776
HF	20,000	FMGFD[3]	48	48	0.852090	0.178508	24,844,766	14,664,124
HF	20,000	FMGFD[3]	400	48	0.435327	0.178676	223,480,084	133,812,776
HF	20,000	2DCS	48	48	95.916826	0.178476	70,268,488	40,687,760
HF	20,000	2DCS	400	48	8.137236	0.178749	669,733,976	396,822,704

Table 11 HF Wide Band Medium Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA Error (µs)***	***FDOA error (Hz)***	***Real Adds***	***Real Mul-tiplies***
HF	20,000	FM	2801	2801	333.138259	0.061644	3,219,480	2,118,192
HF	20,000	FMG[1]	2801	2801	171.796572	0.061644	3,219,864	6,421,296
HF	20,000	FMG[1]	2801	400	0.212516	0.018019	15,194,712	30,168,240
HF	20,000	FMG[2]	2801	2801	2.527392	0.065953	3,219,864	6,421,296
HF	20,000	FMG[2]	2801	400	0.293661	0.012058	15,194,712	30,168,240
HF	20,000	FMGFD[4]	2801	2801	171.373295	0.061655	23,305,856	13,581,568
HF	20,000	FMGFD[4]	2801	400	1.103619	0.010625	201,729,600	119,557,248
HF	20,000	FMGFD[5]	2801	2801	11.362605	0.065900	23,305,856	13,581,568
HF	20,000	FMGFD[5]	2801	400	0.762588	0.010871	201,729,600	119,557,248
HF	20,000	FMGFD[3]	2801	2801	174.374113	0.061878	23,305,856	13,581,568
HF	20,000	FMGFD[3]	2801	400	0.980649	0.010554	201,729,600	119,557,248
HF	20,000	2DCS	2801	2801	186.022185	0.061972	38,252,352	22,503,040
HF	20,000	2DCS	2801	400	9.867400	0.010686	327,326,964	194,427,368

Table 12 HF Wide Band High Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (µs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
VHF	5,000	FM	6	6	0.013189	0.007319	20,076,504	12,335,664
VHF	5,000	FMG[1]	6	6	0.166738	0.007340	20,252,520	16,912,080
VHF	5,000	FMGFD[4]	6	6	1.809923	0.007331	26,431,436	15,773,080
VHF	5,000	FMGFD[3]	6	6	1.896360	0.007329	26,431,436	15,773,080
VHF	5,000	2DCS	6	6	472.101980	0.013532	141,304,060	82,049,784

Table 13 VHF Narrow Band Low Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (µs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
VHF	5,000	FM	12	12	0.000647	0.007128	11,032,536	6,830,640
VHF	5,000	FMG[1]	12	12	12.430606	0.007234	11,120,544	11,231,040
VHF	5,000	FMGFD[4]	12	12	10.579849	0.007169	25,892,698	15,384,628
VHF	5,000	FMGFD[3]	12	12	12.262696	0.007161	25,892,698	153,84,628
VHF	5,000	2DCS	12	12	536.056858	0.008140	67,624,872	39,163,216

Table 14 VHF Narrow Band Medium Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (µs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
VHF	5,000	FM	83	83	1.806011	0.220343	3,933,384	2,554,416
VHF	5,000	FMG[1]	83	83	8.083229	0.221603	3,946,128	6,810,912
VHF	5,000	FMGFD[4]	83	83	7.854864	0.221589	25,588,420	15,058,312
VHF	5,000	FMGFD[3]	83	83	7.169745	0.221698	25,588,420	15,058,312
VHF	5,000	2DCS	83	83	349.286349	0.221732	34,554,672	20,037,216

Table 15 VHF Narrow Band High Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (µs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
VHF	10,000	FM	12	12	0.546796	0.301155	11,032,536	6,830,640
VHF	10,000	FMG[1]	12	12	5.907726	0.301125	11,120,544	11,231,040
VHF	10,000	FMGFD[4]	12	12	4.853693	0.301159	25,892,698	15,384,628
VHF	10,000	FMGFD[3]	12	12	5.685575	0.301164	25,892,698	15,384,628
VHF	10,000	2DCS	12	12	267.971858	0.300257	67,624,872	39,163,216

Table 16 VHF Medium Band Low Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (µs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
VHF	10,000	FM	24	24	1.232829	0.301164	6,806,040	4,274,736
VHF	10,000	FMG[1]	24	24	1.772533	0.300911	6,850,056	8,588,304
VHF	10,000	FMGFD[4]	24	24	1.920431	0.300903	25,366,620	15,019,192
VHF	10,000	FMGFD[3]	24	24	0.127705	0.300841	25,366,620	15,019,192
VHF	10,000	2DCS	24	24	233.948641	0.300885	68,913,712	39,863,904

Table 17 VHF Medium Band Medium Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
VHF	10,000	FM	166	166	14.029722	0.108862	3,519,576	2,308,656
VHF	10,000	FMG[1]	166	166	7.230216	0.109366	3,525,960	6,560,400
VHF	10,000	FMGFD[4]	166	166	7.291491	0.109390	25,257,780	14,835,304
VHF	10,000	FMGFD[3]	166	166	8.201645	0.109351	25,257,780	14,835,304
VHF	10,000	2DCS	166	166	158.616845	0.109118	35,351,316	20,545,064

Table 18 VHF Medium Band High Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
VHF	20,000	FM	24	24	0.242958	0.151487	6,806,040	4,274,736
VHF	20,000	FMG[1]	24	24	0.109891	0.151560	6,850,056	8,588,304
VHF	20,000	FMGFD[4]	24	24	0.126921	0.151559	25,366,620	15,019,192
VHF	20,000	FMGFD[3]	24	24	1.190068	0.151592	25,366,620	15,019,192
VHF	20,000	2DCS	24	24	116.091211	0.151565	68,913,712	39,863,904

Table 19 VHF Wide Band Low Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
VHF	20,000	FM	48	48	0.444009	0.151745	4,839,960	3,095,088
VHF	20,000	FMG[1]	48	48	0.720542	0.151645	4,861,968	7,364,640
VHF	20,000	FMGFD[4]	48	48	0.653187	0.151652	24,844,766	14,664,124
VHF	20,000	FMGFD[3]	48	48	0.612315	0.151666	24,844,766	14,664,124
VHF	20,000	2DCS	48	48	96.019831	0.151576	70,268,488	40,687,760

Table 20 VHF Wide Band Medium Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
VHF	20,000	FM	333	333	3.936305	0.001778	3,525,960	2,308,656
VHF	20,000	FMG[1]	333	333	5.585045	0.000779	3,529,152	6,566,784
VHF	20,000	FMGFD[4]	333	333	5.569979	0.000857	41,164,064	24,042,048
VHF	20,000	FMGFD[3]	333	333	7.218809	0.000776	41,164,064	24,042,048
VHF	20,000	2DCS	333	333	64.265958	0.000459	67,061,472	38,882,752

Table 21 VHF Wide Band High Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (µs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
UHF	5,000	FM	6	6	0.419396	0.224765	20,076,504	12,335,664
UHF	5,000	FMG[1]	6	6	0.671609	0.224621	20,252,520	16,912,080
UHF	5,000	FMGFD[4]	6	6	1.245058	0.224350	26,431,436	15,773,080

UHF	5,000	FMGFD[3]	6	6	1.313849	0.224241	26,431,436	15,773,080
UHF	5,000	2DCS	6	6	477.922349	0.218929	141,304,060	82,049,784

Table 22 UHF Narrow Band Low Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (μs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
UHF	5,000	FM	7	7	0.635383	0.209007	11,032,440	6,830,640
UHF	5,000	FMG[1]	7	7	3.563571	0.208471	11,183,304	11,356,560
UHF	5,000	FMGFD[4]	7	7	2.065287	0.208663	13,829,944	8,152,176
UHF	5,000	FMGFD[3]	7	7	2.165415	0.208630	13,829,944	8,152,176
UHF	5,000	2DCS	7	7	522.907829	0.206698	73,809,616	42,820,000

Table 23 UHF Narrow Band Medium Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (μs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
UHF	5,000	FM	8	8	0.150217	0.240312	11,032,344	6,830,640
UHF	5,000	FMG[1]	8	8	2.172152	0.239282	11,164,344	11,318,640
UHF	5,000	FMGFD[4]	8	8	0.403180	0.240375	13,422,792	7,941,520
UHF	5,000	FMGFD[3]	8	8	1.258024	0.240476	13,422,792	7,941,520
UHF	5,000	2DCS	8	8	543.668976	0.243241	71,950,752	41,714,752

Table 24 UHF Narrow Band High Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (μs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
UHF	10,000	FM	12	12	0.052927	0.109282	11,032,536	6,830,640
UHF	10,000	FMG[1]	12	12	6.235831	0.110565	11,120,544	11,231,040
UHF	10,000	FMGFD[4]	12	12	5.189911	0.109908	25,892,698	15,384,628
UHF	10,000	FMGFD[3]	12	12	5.895590	0.109808	25,892,698	15,384,628
UHF	10,000	2DCS	12	12	268.733374	0.109960	67,624,872	39,163,216

Table 25 UHF Medium Band Low Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (μs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
UHF	10,000	FM	14	14	0.822300	0.163491	6,805,368	4,274,736
UHF	10,000	FMG[1]	14	14	2.857153	0.163854	6,880,800	8,649,792
UHF	10,000	FMGFD[4]	14	14	2.222264	0.163833	13,929,870	8,193,692
UHF	10,000	FMGFD[3]	14	14	2.098840	0.163774	13,929,870	8,193,692
UHF	10,000	2DCS	14	14	269.279843	0.167318	35,432,552	20,562,640

Table 26 UHF Medium Band Medium Decimation Test Case Results

Frequency	*Bandwidth (Hz)*	*Method*	*L*	*D*	*TDOA error (μs)*	*FDOA error (Hz)*	*Real Adds*	*Real Multiplies*
UHF	10,000	FM	18	18	1.157910	0.064482	6,805,752	4,274,736

UHF	10,000	FMG[1]	18	18	1.586039	0.063606	6,864,432	8,617,056
UHF	10,000	FMGFD[4]	18	18	0.853834	0.062767	27,325,474	16,121,412
UHF	10,000	FMGFD[3]	18	18	0.175996	0.062676	27,325,474	16,121,412
UHF	10,000	2DCS	18	18	266.148526	0.065335	72,227,776	41,799,552

Table 27 UHF Medium Band High Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (μs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
UHF	20,000	FM	24	24	0.983472	0.247195	6,806,040	4,274,736
UHF	20,000	FMG[1]	24	24	0.681083	0.245018	6,850,056	8,588,304
UHF	20,000	FMGFD[4]	24	24	0.450447	0.244974	25,366,620	15,019,192
UHF	20,000	FMGFD[3]	24	24	1.500222	0.244327	25,366,620	15,019,192
UHF	20,000	2DCS	24	24	115.251124	0.244812	68,913,712	39,863,904

Table 28 UHF Wide Band Low Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (μs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
UHF	20,000	FM	28	28	0.355426	0.183001	6,806,040	4,274,736
UHF	20,000	FMG[1]	28	28	0.957560	0.185293	6,843,768	8,575,728
UHF	20,000	FMGFD[4]	28	28	1.059172	0.185607	24,526,648	14,546,544
UHF	20,000	FMGFD[3]	28	28	1.033181	0.185804	24,526,648	14,546,544
UHF	20,000	2DCS	28	28	112.199927	0.185861	67,490,508	390,29,656

Table 29 UHF Wide Band Medium Decimation Test Case Results

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error (μs)***	***FDOA error (Hz)***	***Real Adds***	***Real Multiplies***
UHF	20,000	FM	33	33	0.958216	0.112369	6,806,328	4,274,736
UHF	20,000	FMG[1]	33	33	1.142923	0.111754	6,838,344	8,564,880
UHF	20,000	FMGFD[4]	33	33	0.670270	0.111178	48,967,984	29,066,848
UHF	20,000	FMGFD[3]	33	33	1.258652	0.111416	48,967,984	29,066,848
UHF	20,000	2DCS	33	33	106.784758	0.112195	66,196,364	38,269,208

Table 30 UHF Wide Band High Decimation Test Case Results

Notes:
FB = Filter Bank
FM = Fine Mode
FMG = Fine Mode Generic
FMGFD = Fine Mode Generic Frequency Domain
2DCS = 2-D Cross Spectra
[1] Filter used for test case was designed using Matlab fir1 command
[2] Filter used for test case was designed using Matlab Filter Design Tool
[3] Window used for test case was a Tukey window
[4] Filter used for test case was same one as for FMG
[5] Filter designed using Matlab Filter Design Tool, same as for FMG
🟨 Best accuracy for *L*=*D* only

🟩 Best accuracy for both *L*=*D* and *L* > *D*

🟦 When Best accuracy when *L*=*D* is the same as for both *L*=*D* and *L* > *D*

As noted in Chapter 3 and from the results displayed in Table 4 through Table 12, accuracy tends to improve for methods when *L*>*D* as opposed to when *L*=*D*. However from the results obtained from the test cases, it is clear that filter/window used plays an important role in determining the accuracy of the correlation. An observation made from the results shows that what were considered "better" filters/windows from Chapter 4 Section 4.2 in some of the HF test cases were not the "better" filters/windows after all. Some of the other filters/windows performed significantly better, in some of the test cases.

In terms of accuracy regarding the methods with data reduction (e.g. decimation) there was no clear-cut best method. However, the 2-D Cross Spectra method performed the poorest in TDOA accuracy. TDOA accuracy improved for this method when *L*>*D*, but it was still the worst method in TDOA accuracy. It is unclear as to why. An analysis was performed on select cases using this method to verify the correctness of the coding of the algorithm. A visual examination of the TDOA dimension of the CAF surface showed that in all the cases, the surface looks fine, but upon zooming in around the peak, the peak is not close to the expected value of 0 ns. From this information, it was believed that the Matlab fftshift applied to the DFT in order to center the DFT at zero frequency was re-ordering the data incorrectly and thus creating an incorrect surface. This hypothesis was tested by examining the code and running a test where DFTs followed by the fftshifts were taken on the individual columns of the *CS* and rows of the A_{Doppler} matrices instead of the matrices themselves, the same results were produced as when DFTs followed by fftshifts were taken on the entire matrices. Further study is suggested to determine why the TDOA accuracy is grossly different from the other methods. It is believed that the method has been coded correctly in Matlab.

It is clear from Table 4 - Table 30 that the results are dependent upon the bandwidth over the sampling rate. In order to put the results from Table 4 - Table 30 into perspective, the Cramer-Rao lower bounds (CRLB) were computed for TDOA and FDOA for each of the test frequencies (HF, VHF, and UHF). The plots for the CRLB for TDOA and FDOA are shown in Figure 39 and Figure 40, respectively.

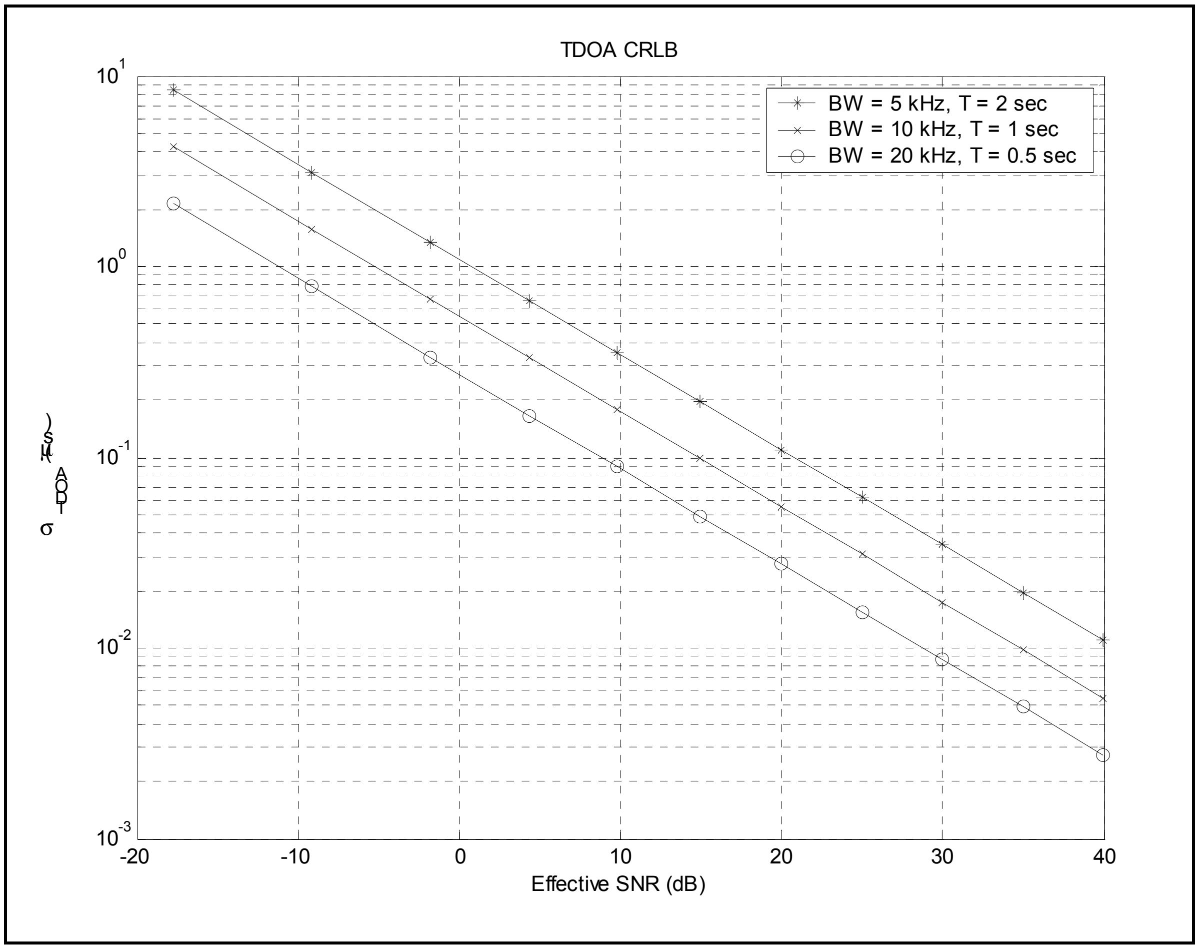

Figure 39 Cramer-Rao Lower Bounds for TDOA

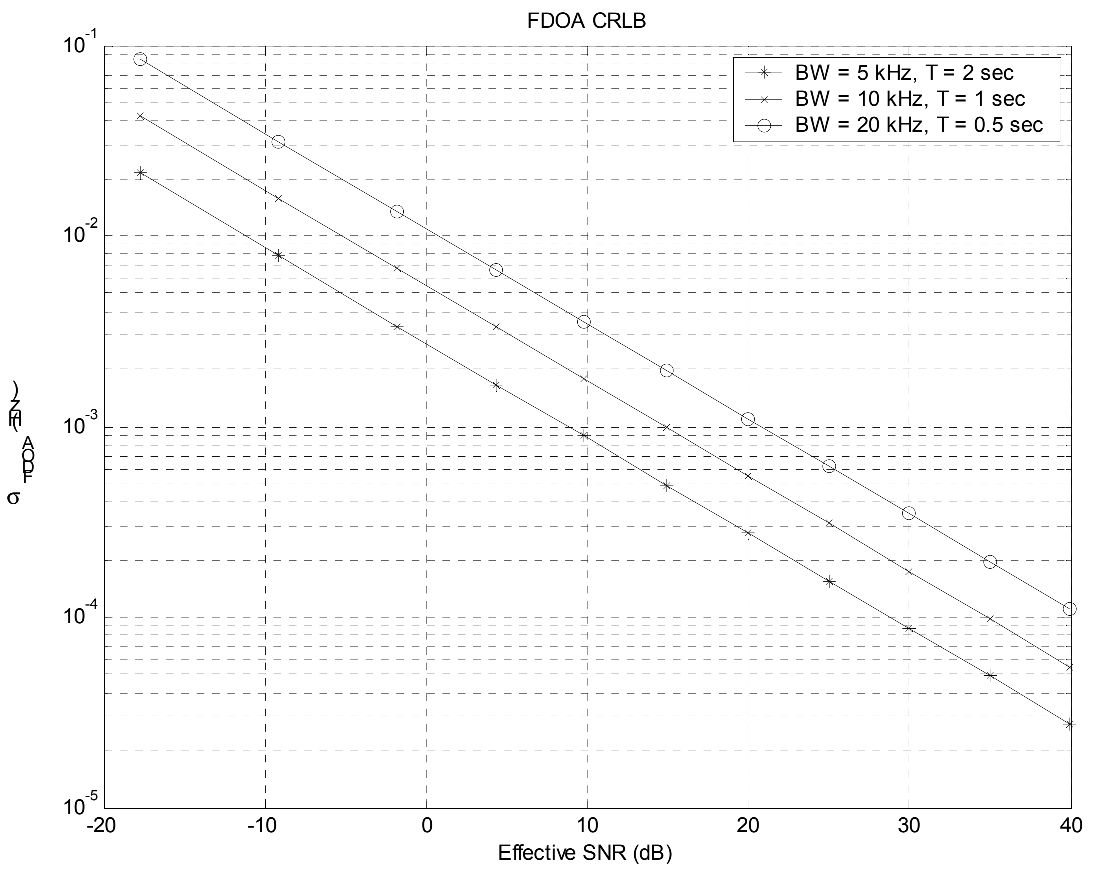

Figure 40 Cramer-Rao Lower Bounds for FDOA

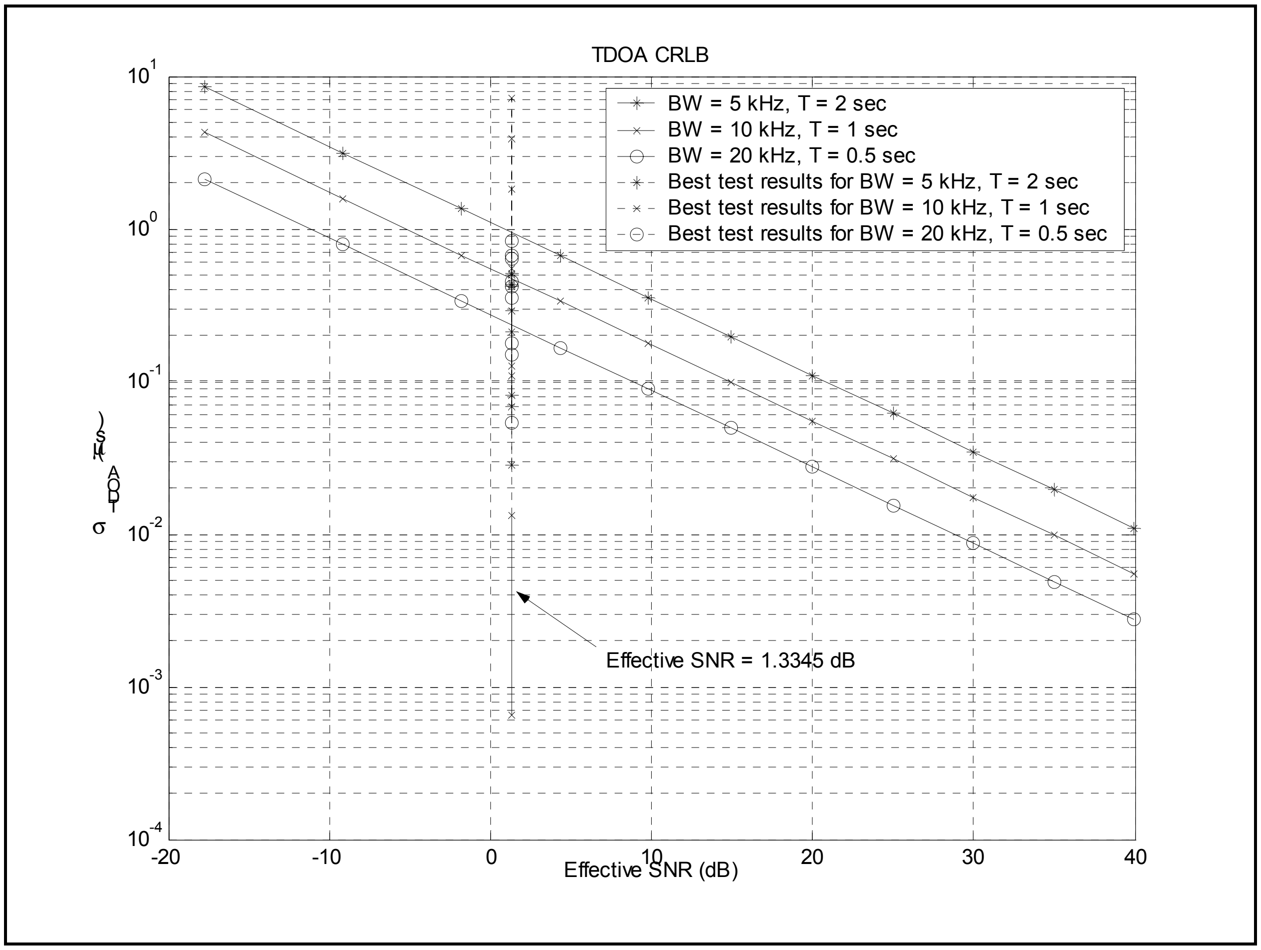

Figure 41 Cramer-Rao Lower Bounds for TDOA with Best Test Result Values from Tables 4 -30

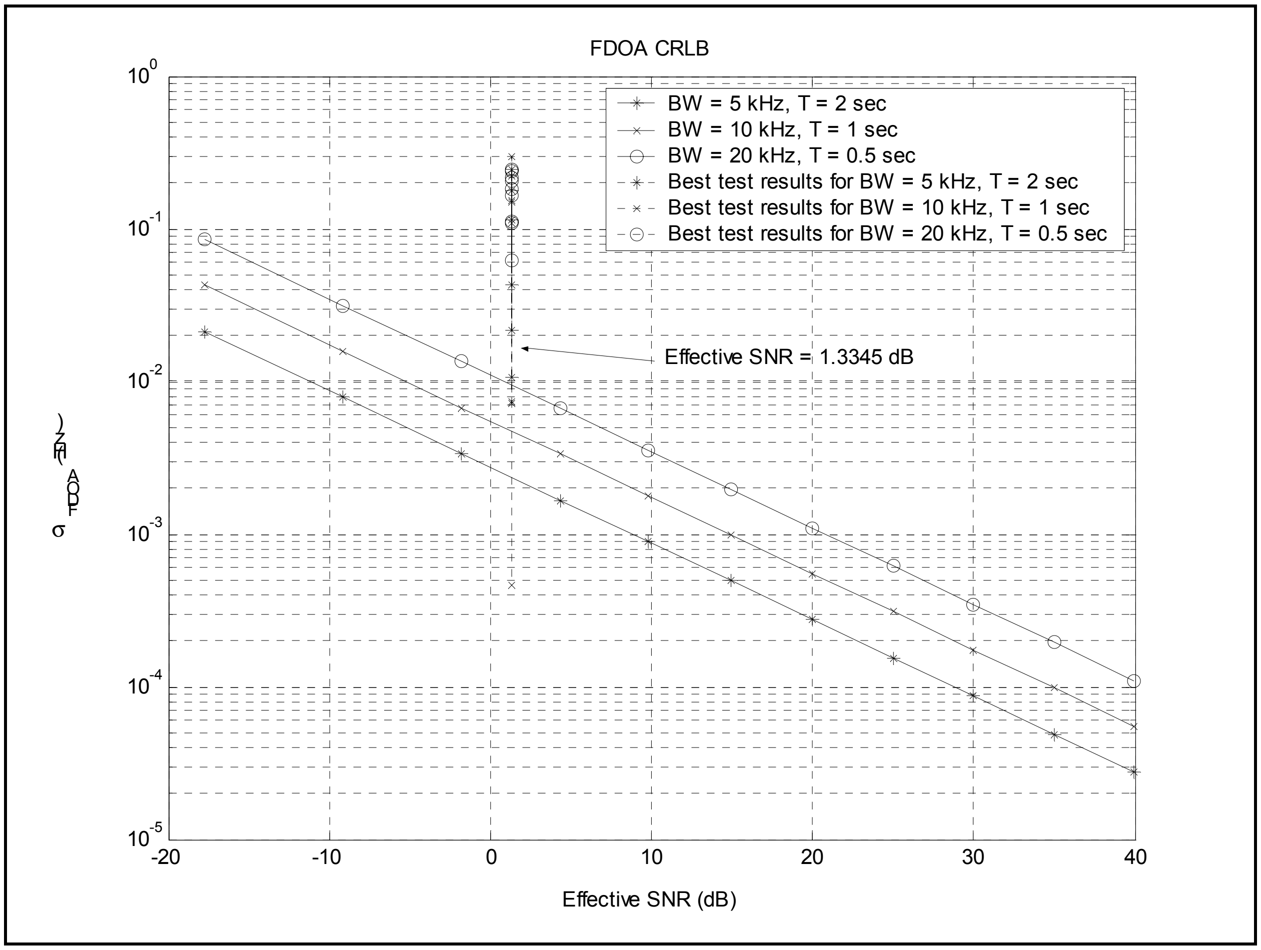

Figure 42 Cramer-Rao Lower Bound for FDOA with Best Test Result Values from Tables 4 - 30

The equations used to compute the CRLB were 7 and 8 from [1]. When calculating the CRLB the noise bandwidth was assumed to be flat (rectangular) and was set equal to the signal bandwidth and the assumptions made in equations 11b and 12b from [1] were also applied. Equation 13 from [1] was used to calculate the effective signal-to-noise-ratio (SNR). As mentioned in Chapter 4 Section 4.2, the SNR on each stream was assumed to be the same since no fading effects were simulated on the data. Therefore, when calculating the CRLBs, only cases where $\gamma_1=\gamma_2$ were considered. Since the CRLB is a measure of the best possible accuracy, from comparing the results obtained from Table 4 - Table 30 and the CRLB from Figure 39 and Figure 40, it is clear that the results obtained best match the effective SNR range from $\leq$ 15 dB. Given that there was no noise added to the test signals and that there was ~40 dB of processing gain, the results obtained for the data reduction methods are not as expected. The highlighted best values obtained from the test results for TDOA and FDOA from Table 4 - Table 30 are superimposed on the CRLB plots in Figure 41- Figure 42. Some values are below the CRLB. Given that there was no noise present, this is expected. However, these are just the best values. All of the test result values should be at or below the CRLBs. The effective SNR calculated from the data was approximately 1.33 dB. The effective SNR was calculated by taking the average power of both signal streams and then applying them to equation 13 from [1].

The results obtained from the Filter Bank method are better than the CRLB results, as expected. The majority of the results obtained from the data reduction methods are not. For the data reduction methods, with the exception of the 2-D Cross Spectra method, visual confirmation of the CAF surfaces look reasonable and upon zooming in around the peak of the CAF surfaces, many cases look are good, but not as good as is expected with no noise! After the analysis, no errors were determined in relation to what was expected. As with the 2-D Cross Spectra method, it is believed that the methods were coded correctly in Matlab. Further analysis is suggested to determine why the results are not better than what was obtained.

Another observation made for all the cases with data reduction is that when *D* is particularly large (> 100), TDOA accuracy for all the methods becomes poorer than when *D* is < 100. This is more evident in the HF cases when the decimation, *D* goes above 100. However for the VHF test cases, the TDOA error is on the order of 5-10 times worse for the high decimation levels, compared to the medium decimation rates. This is not so evident in the UHF cases, most likely because of the decimation rates do not go above 100 because they are restricted to be ≤ the filter length, *L*. To examine this phenomenon the HF medium bandwidth test case with L=D was examined. The case for the "Fine-Mode" was selected because since all the other methods are based off of the "Fine-Mode" method, if any method was to provide insight into what was happening, the "Fine-Mode" method would be the one. The lag product for the first TDOA shift was examined. This lag product was plotted in the frequency domain with the "all-ones" filter and its frequency shifted replicas super-imposed. This was done for each filter length and decimation for this test case (*L*=*D*=12, *L*=*D*=24, *L*=*D*=1400). The plots are in Figure 43 through Figure 45. As can be seen from Figure 43 through Figure 45, there is aliasing for the low and the medium decimation rates. The aliasing that occurs in for these two rates does not effect the Doppler range of interest (-5 Hz to 5 Hz). Though it can not be seen from Figure 45, there is aliasing present. A zoomed in view is available in Figure 46. The Aliasing for the high decimation rates does occur over the Doppler range of interest and seems to be the major contributing factor to the poor accuracy for the high decimation rates (> 100).

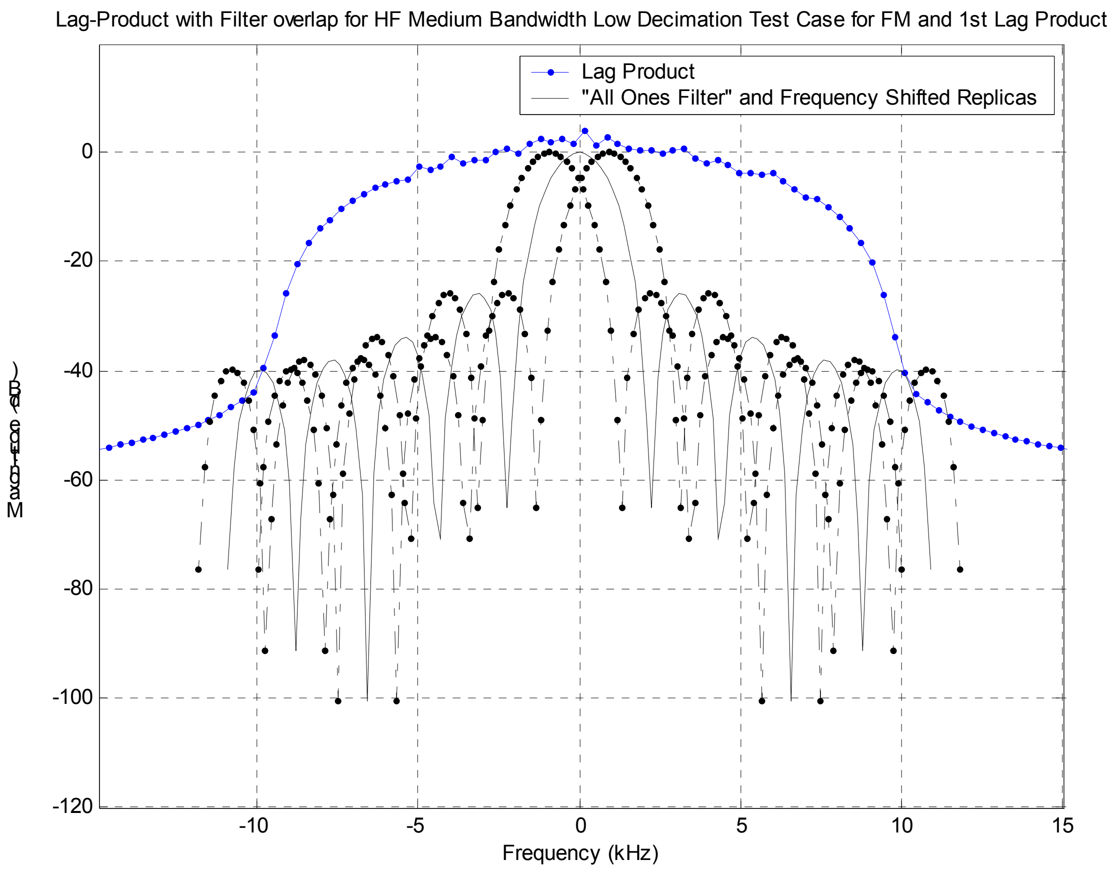

Figure 43 PSD of Lag Product for Filter Low Decimation, FM Method, DFT Size = 128

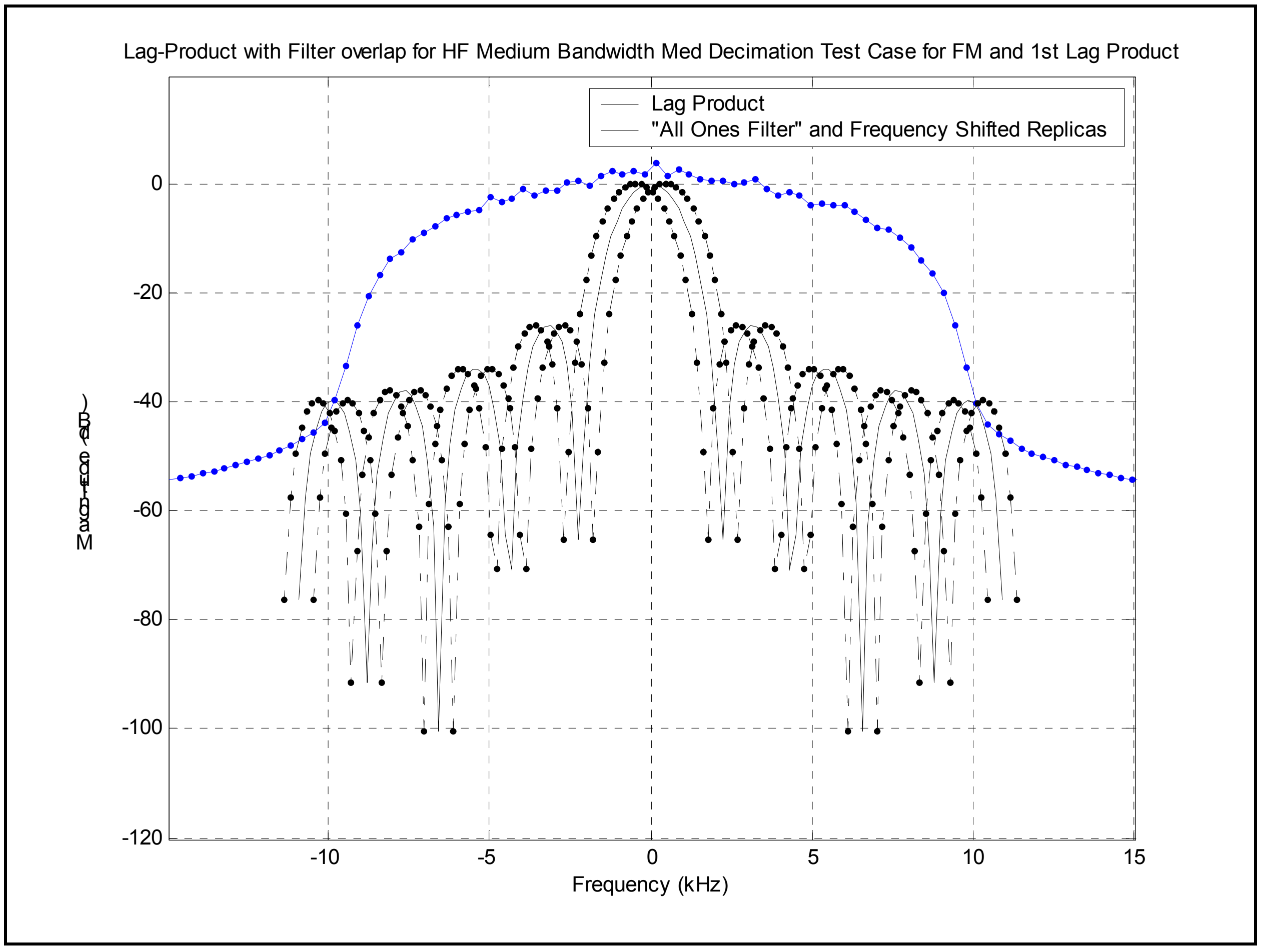

Figure 44 PSD of Lag Product with Filter Medium Decimation, FM Method DFT Size = 128

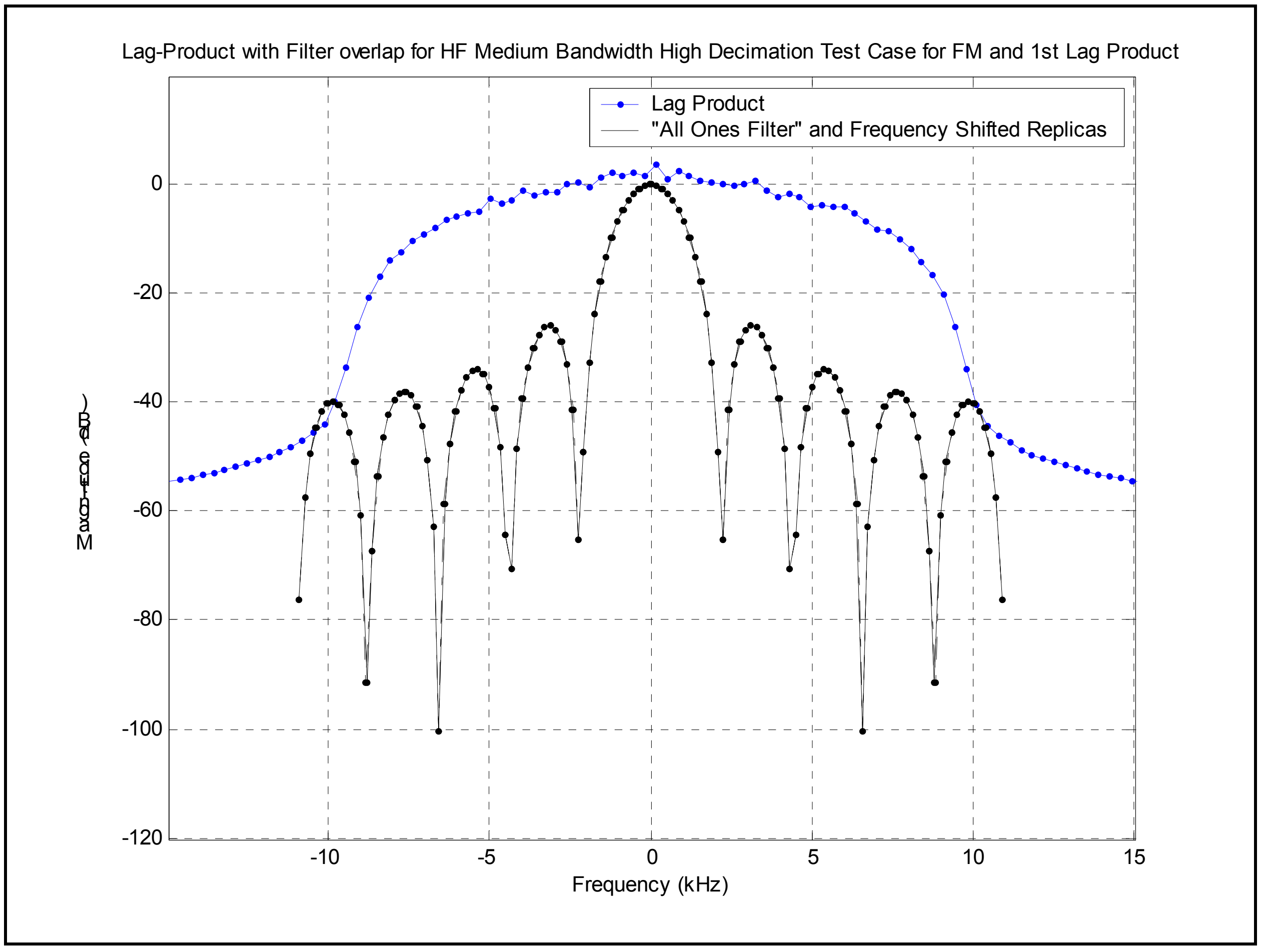

Figure 45 PSD of Lag Product with Filter High Decimation, FM Method, DFT Size = 128

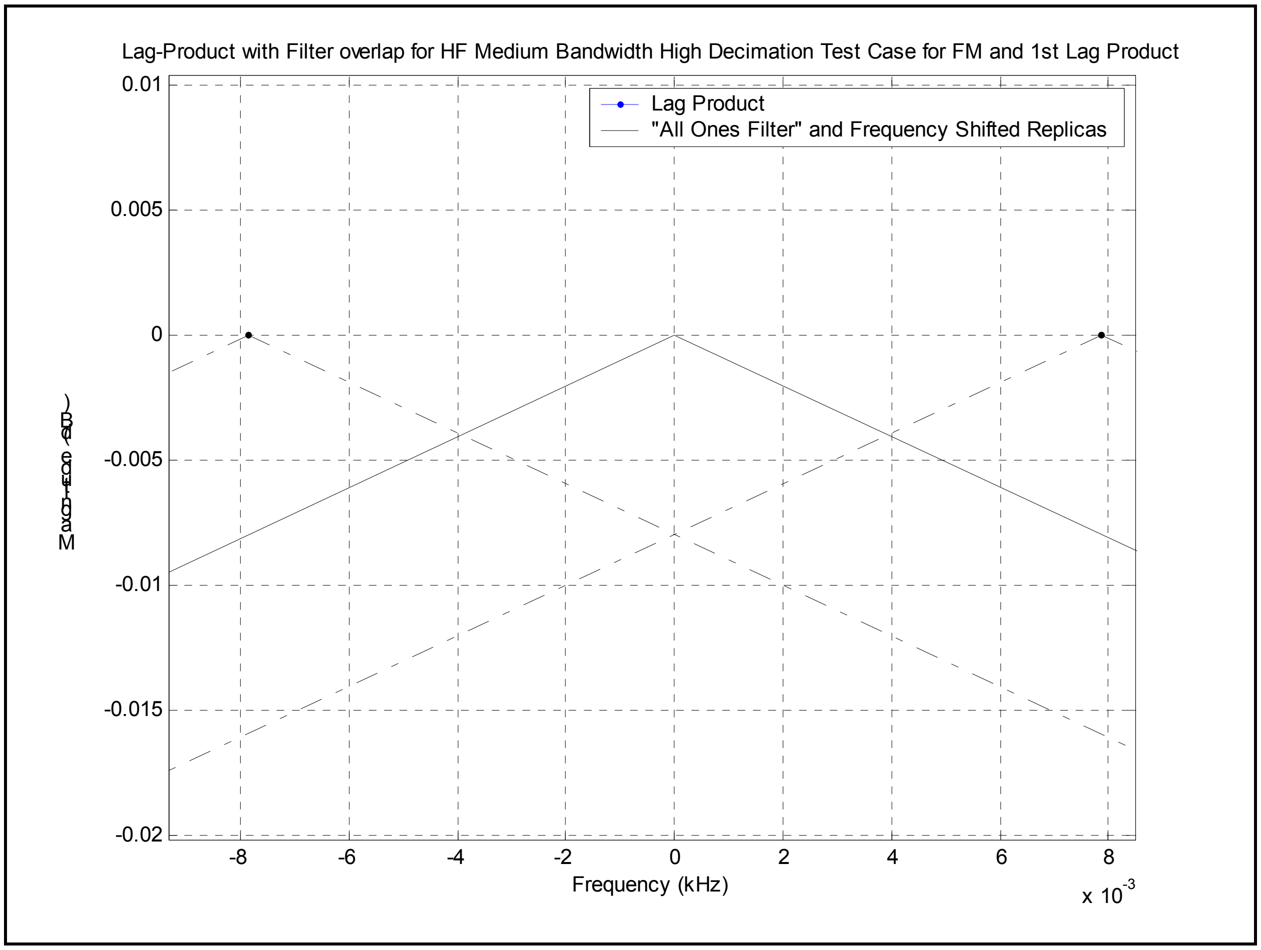

Figure 46 Zoomed in View of Aliasing for HF Med BW, High Decimation Test Case

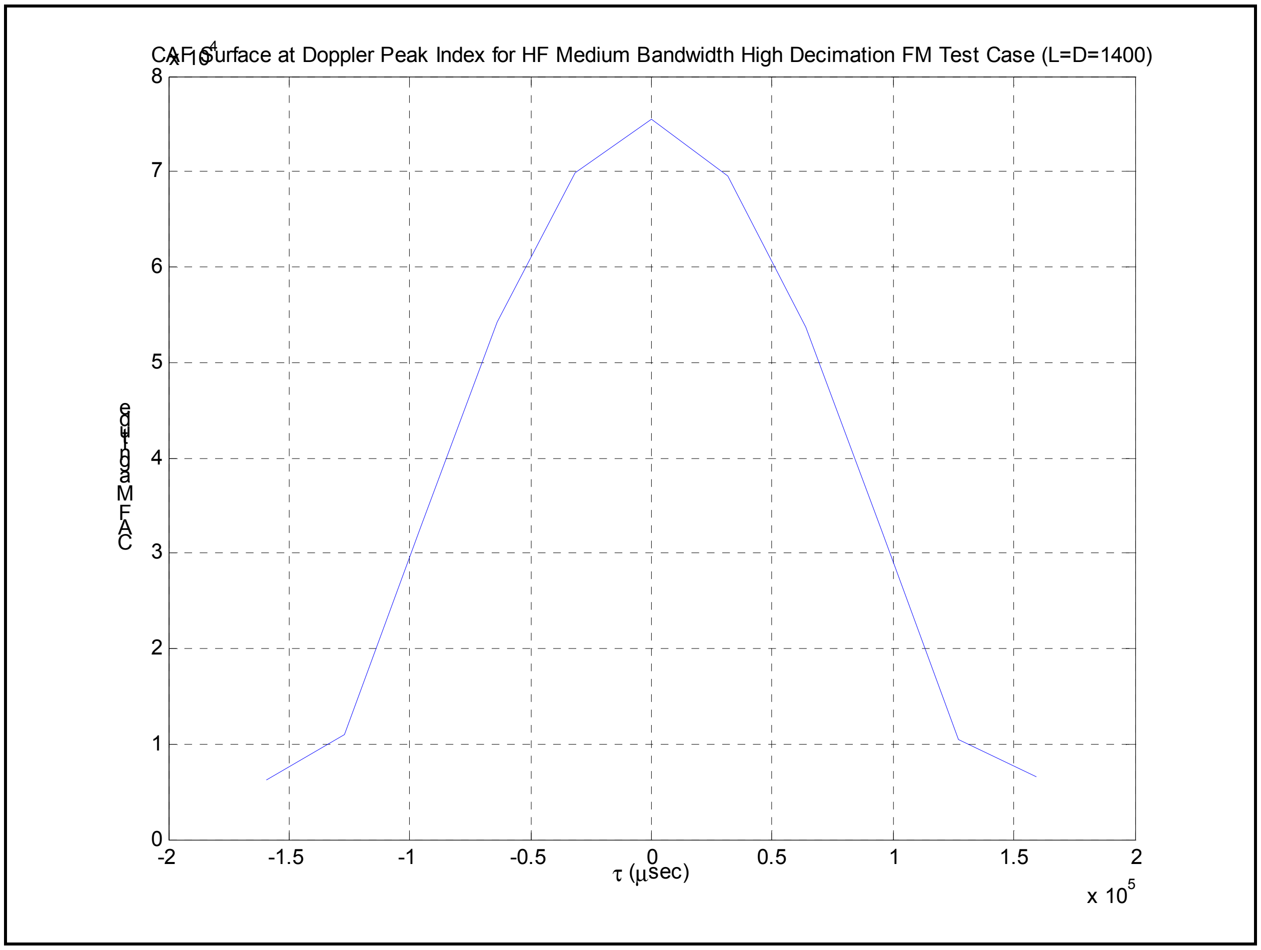

Figure 47 CAF Plot in TDOA Dimension for HF Medium Band High Decimation FM Test Case

In order to see what effect the aliasing had on the surface, the CAF in the TDOA dimension was plotted for the Doppler value that resulted in the highest peak in the Doppler dimension. This plot is in Figure 47. The CAF in the TDOA dimension looks like it is centered at 0 micro-seconds. However, after the curve-fit is applied, the TDOA is computed to be 279.987817 micro-seconds (see Table 9). It appears that though there is aliasing, it is not seem to affect the CAF much at all. In fact it is not the major error but curve-fit error that causes the large TDOA error. However, after examining x-axis in Figure 47, it was determined that the x-axis is in tens of micro-seconds. To get an accurate TDOA estimate, there should be five points around 0 micro-seconds. This indicates that the Tau-spacing used for this test case is too coarse. By increasing the sampling rate, TDOA accuracy will improve. This test case was executed again with increased sampling rates (better Tau-spacing), and the results are in.Table 31. With each running, the TDOA results did improve. Since for all the high decimation rates over 100, the accuracy degraded significantly, large oversampling rates appear to be required. Further study is needed to verify this result and to determine an optimal oversampling value that will yield a balance between accuracy and computational complexity.

Frequency (MHz)	*Sampling Rate (kHz)*	*L*	*D*	*TDOA Error (μs)*	*FDOA Error (Hz)*	*Real Adds*	*Real Mults*
5	11	1400	1400	1159.855401	0.036731	3,211,416	2,114,352
5	44	1400	1400	279.987817	0.125448	3,235,608	2,127,408
5	110	1400	1400	111.995127	0.264527	3,274,008	2,148,912
5	220	1400	1400	49.892726	0.096647	3,360,024	2,198,064
5	330	1400	1400	33.261817	0.194150	3,550,488	2,308,656
5	440	1400	1400	24.946363	0.194150	3,550,488	2,308,656
5	550	1400	1400	19.957090	0.194150	3,550,488	2,308,656
5	110000	1400	1400	0.101601	0.198817	166,784,280	102,775,344

Table 31 Results From Increased Oversampling

Examining the results in Table 4 through Table 30, the “Fine-Mode” Generic and “Fine-Mode” Generic Frequency Domain methods when using the same filter/window were about the same in accuracy for the majority of the test cases. This makes sense because the implementation for both methods is the same, except one is implemented in the time domain and the other in the frequency domain [2]. The FDOA accuracy discrepancy between both methods based on Table 4 through Table 30 is on the order of ~ 1 MHz. This is quite good accuracy and support the claim made in [2] that these two methods produce the same results. Most of the test case results display TDOA accuracy discrepancies between these two methods of ~1us. Notable exceptions are the HF medium and wide bandwidth high decimation test cases with *L*=*D* and using the filters designed using the Matlab filter design tool (Table 9 and Table 12). A contributing factor to this discrepancy was discovered to be because of curve-fit error. The Matlab polyfit command was used to perform the curve-fit. Upon examining the cross-ambiguity function (CAF) for the TDOA curve-fit for the HF wide bandwidth high decimation test case from Table 12, the values for both the methods were the same to 0.01 numerical precision, yet two different curve-fits were produced and thus two different TDOA values. The CAF values used for the curve-fit for both methods are shown in Table 32.

FMG Method ***CAF Values***	***FMGFD Method*** ***CAF Values***
20.16231456739720	20.18102705010027
25.91042656222847	25.93185268724099
28.01950169363270	28.01429643250628
25.91145176432027	25.92680221576133
20.16334521420673	20.17662377051057

Table 32 CAF Values for TDOA Curve Fit for HF Wide Bandwidth High Decimation Test

For the HF medium bandwidth high decimation test case from Table 9, the CAF values between the two methods for the TDOA curve-fit were off by about 1. The discrepancy between the two methods was larger than the one for the HF wide bandwidth high decimation test case, but given the values it was expected that the results would be more similar. The CAF values used for the curve-fit for both methods for this test case are shown in Table 33.

FMG Method ***CAF Values***	***FMGFD Method*** ***CAF Values***
39.18582718831557	40.78948949752790
50.83887011809706	52.69312488560406
55.10254091606012	55.14852592401184
50.82277097553202	52.49065539932252
39.19567053020109	40.48488978636797

Table 33 CAF Values for TDOA Curve Fit for HF Medium Bandwidth High Decimation Test Case

Upon further review, the curve-fit for the HF medium bandwidth high decimation test case from Table 9 was plotted using the "Fine-Mode" Generic Frequency Domain method. Another curve-fit plot was super-imposed on the first plot using the same CAF values but with several microseconds of perturbation in the time-delay values. The curve-fit was not noticeably changed. It was concluded that the curve-fit errors were in a relative sense rather than an absolute sense. This is analogous to a 20,000 foot view where

10,000 cars are lined up in a row. If 2,000 of these cars are moved one foot forward, the cars will still appear to be lined up in a row.

After examining several other test cases with the TDOA accuracy discrepancies between the “Fine-Mode” Generic and “Fine-Mode” Generic Frequency Domain methods, the curve-fit error was again concluded to be the major contributing factor for the discrepancy. It is not known why the CAF values are not identical. It was hypothesized that the “Fine-Mode” Generic Frequency Domain method had errors because since the number of TDOAs for this method was based on the DFT size and were always the next highest power of two (and in some cases much larger than the number of TDOAs for the “Fine-Mode” Generic method) many small errors could be adding up over time and result in a large TDOA error. This was disproved when the minimum DFT size for TDOA was used. The same exact TDOA value resulted. It was also thought that not enough TDOAs were being used and resulting in a poor surface in the TDOA direction and that by adding more TDOAs accuracy would improve. This theory was also disproved because when using a larger DFT size, again the same TDOA resulted. This made sense in that the TDOA computed for the minimum DFT size, the normal DFT size based, and the larger DFT size all produced the same result. The normal and larger DFT size only had more zeros to deal with during the computation than the minimum DFT size. Ultimately, these zeros would be thrown away and the same TDOA result will be produced, which it was. Further analysis is needed to determine why the CAF values for the “Fine-Mode” Generic and “Fine-Mode” Generic Frequency Domain methods were not identical in the TDOA direction of the CAF.

Tolimieri and Winograd in [2] claim that the “Fine-Mode” Generic Frequency Domain method is more computationally efficient than the “Fine-Mode” Generic method when L is sufficiently large. From examining the number of computations in Table 4 through Table 30, it is clear that was not the case, especially when L was large (L=1400 in Table 9 and L = 2801 in Table 12). After analysis, it was concluded that perhaps the zero-padding used for the “Fine-Mode” Generic Frequency Domain method (zero-padding to $2L$-1 so as to get linear correlation instead of circular correlation [see Chapter 3 Section 3.1]) was more than what was needed. In [2], Tolimieri and Winograd propose a method for obtaining linear correlation via circular correlation. Instead of zero-padding each correlation input block to 2L-1, the first correlation input block ($f_m[n]$) is zero padded to a length L+T, where T is the number of time-delays. The second correlation input block ($s_2[n]$) is always, L+T samples long. By not zero-padding to a length 2L-1 and taking a DFT of length 2L-1, the number of computations needed to compute the "Fine-Mode" Frequency Domain method should be reduced to a level such that it is computationally efficient. This was tested for all the cases that used the fir1 filter and the results appear in Table 34. Comparing the results from Table 4 through Table 30 with the results in Table 34, it is clear that the larger amount of zero-padding as is consistent with the frequency domain implementation of filters has much better accuracy. The computational complexity is about the same for both zero-pad implementations.

Frequency	***Bandwidth (Hz)***	***Method***	***L***	***D***	***TDOA error***	***FDOA error***	***Real Adds***	***Real Multiplies***
HF	5,000	FMGFD	6	6	41.037224	0.113341	54,843,052	32,690,264
HF	5,000	FMGFD	100	6	0.880685	0.113372	237128372	143915368
HF	5,000	FMGFD	12	12	13.423914	0.113327	25,892,698	15,384,628
HF	5,000	FMGFD	100	12	1.836173	0.113367	112,280,616	67,768,400
HF	5,000	FMGFD	700	700	69.164339	0.262675	11,405,826	6,681,604
HF	5,000	FMGFD	700	100	1.449117	0.043614	103,843,740	62,082,872
HF	10,000	FMGFD	12	12	7.248036	0.245023	25,892,698	15,384,628
HF	10,000	FMGFD	200	12	0.961954	0.245094	232,811,586	141,015,172
HF	10,000	FMGFD	24	24	11.629854	0.244922	25,366,620	15,019,192
HF	10,000	FMGFD	200	24	1.946362	0.245086	110,123,500	66,319,320
HF	10,000	FMGFD	1400	1400	208.842095	0.125516	11,300,544	6,610,304
HF	10,000	FMGFD	1400	200	1.145049	0.021616	100,960,676	60,183,368
HF	20,000	FMGFD	24	24	4.784969	0.178514	25,366,620	15,019,192
HF	20,000	FMGFD	400	24	0.069797	0.178701	228,328,412	138,024,888
HF	20,000	FMGFD	48	48	3.639283	0.178512	11,940,958	7,073,468
HF	20,000	FMGFD	400	48	0.090013	0.178679	107,913,492	64,844,328
HF	20,000	FMGFD	2801	2801	175.245943	0.061655	11,107,968	6,487,296
HF	20,000	FMGFD	2801	400	1.576495	0.010625	97,322,560	57,805,952
VHF	5,000	FMGFD	6	6	41.146000	0.007331	54,843,052	32,690,264
VHF	5,000	FMGFD	12	12	13.742184	0.007169	25,892,698	15,384,628
VHF	5,000	FMGFD	83	83	28.049687	0.221589	12,228,164	7,212,680
VHF	10,000	FMGFD	12	12	7.334231	0.301159	25,892,698	15,384,628
VHF	10,000	FMGFD	24	24	11.918662	0.300903	25,366,620	15,019,192
VHF	10,000	FMGFD	166	166	18.209960	0.109390	12,060,980	7,098,984
VHF	20,000	FMGFD	24	24	4.854528	0.151559	25,366,620	15,019,192
VHF	20,000	FMGFD	48	48	3.856381	0.151652	11,940,958	7,073,468
VHF	20,000	FMGFD	333	333	1.400821	0.000857	20,013,856	11,701,824
UHF	5,000	FMGFD	6	6	42.140362	0.224350	54,843,052	32,690,264
UHF	5,000	FMGFD	7	7	39.963089	0.208663	29,344,536	17,259,824
UHF	5,000	FMGFD	8	8	32.386288	0.240375	28,308,584	16,697,040

UHF	10,000	FMGFD	12	12	6.651027	0.109908	25,892,698	15,384,628
UHF	10,000	FMGFD	14	14	8.864003	0.163833	13,929,870	8,193,692
UHF	10,000	FMGFD	18	18	10.792366	0.062767	13,009,922	7,693,956
UHF	20,000	FMGFD	24	24	4.281872	0.244974	25,366,620	15,019,192
UHF	20,000	FMGFD	28	28	5.747670	0.185607	24,526,648	14,546,544
UHF	20,000	FMGFD	33	33	5.228732	0.111178	23,763,632	14,117,216

Table 34 FMGFD Results with fir1 filter, L=D, and Reduced Zero-Padding

An analysis was also conducted to determine why the "Fine-Mode" Generic Frequency Domain method was not as computationally efficient as the "Fine-Mode" Generic method. As mentioned previously, Tolimieri and Winograd in [2] mentioned for large L, the "Fine-Mode" Generic Frequency Domain method is more computationally efficient. This analysis was performed from a different persepective. In [2], Tolimieri and Winograd have large number of Taus; in some cases the number of Taus is equal to the number of signal samples. For many applications, that is probably far too many time-delays to process. However, for this analysis, the decimation (*D*), filter length (*L*), DFT size for the Doppler calculations were fixed with values representative of what was used in the test cases conducted. The hypothesis was that for not only must the filter length be sufficiently larger than the decimation, but also the number of Taus must be sufficiently large as well. A plot was made to test this hypothesis and is show in Figure

48.

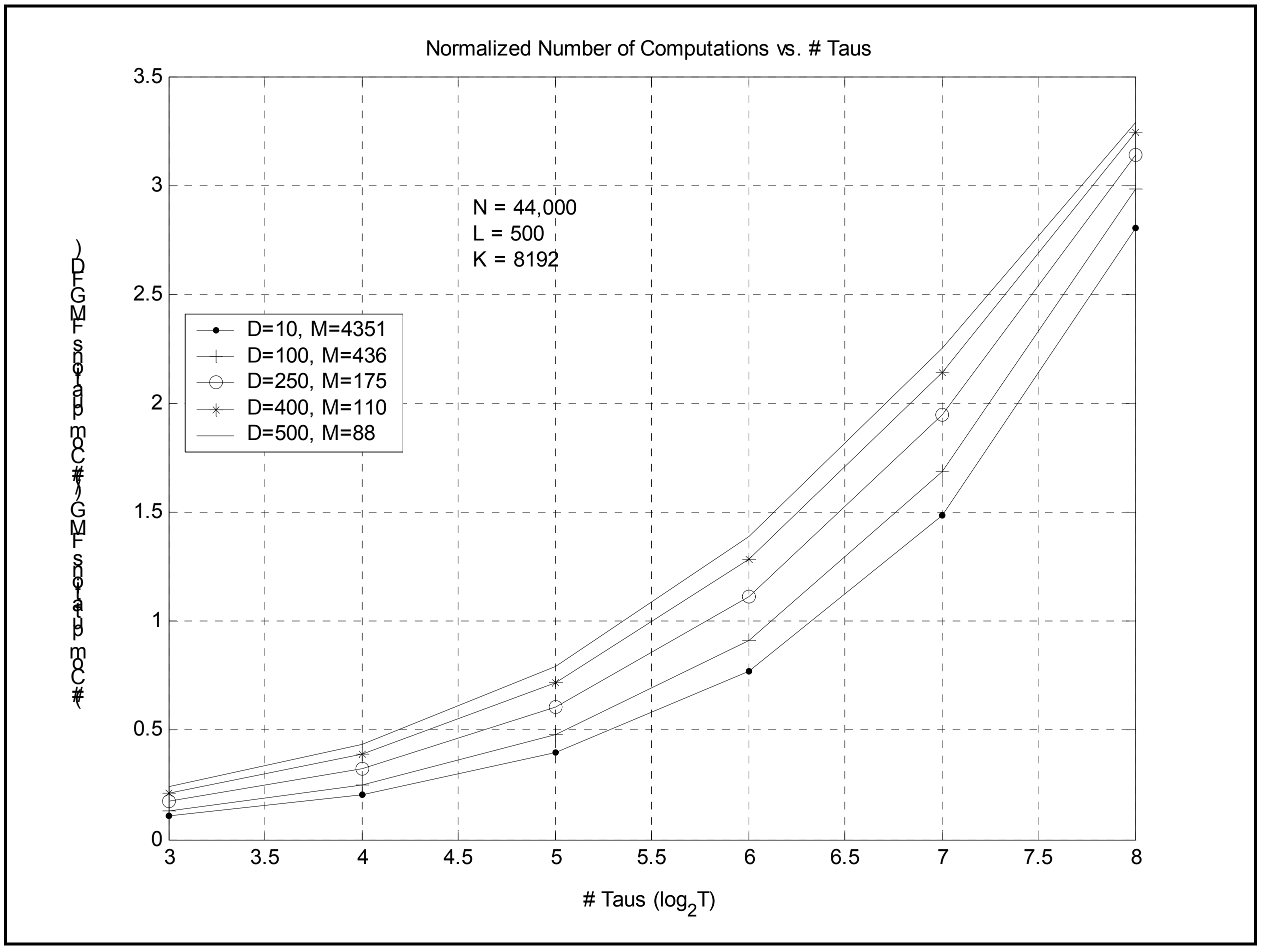

Figure 48 Plot Comparing Computational Complexity of FMG and FMGFD

For the plot in Figure 48, the number of complex computations for the "Fine-Mode" Generic (denoted FMG in Figure 48) and "Fine-Mode" Generic Frequency Domain (denoted FMGFD in Figure 48) were used. For the "Fine-Mode" Generic Frequency Domain method the equation for the number of complex computations used was

$$FMGFD_{comp} = M(L + 3DFT(2L-1) + 2L - 1) + (T+1)(DFT(M) + 2K + 1), \tag{86}$$

where *M* is the number of blocks as computed in (46), *L* is the decimation filter length, *K* is the number of Dopplers, and *T* is the number of Taus. The number of computations used for DFT was taken as the sum of equations 5.23a and 5.23b from Section 5.3 of [9]. The number of complex computations used for the "Fine-Mode" Generic method was

$$FMG_{comp} = (T+1)(N+ML+DFT(M)+2K+1)\,, \tag{87}$$

where , again, *M* is the number of blocks as computed in (46), *L* is the decimation filter length, *K* is the number of Dopplers, and *T* is the number of Taus, and *N* is the number of signal samples. As for the "Fine-Mode" Frequency Domain method computations, number of computations used for DFT was taken as the sum of equations 5.23a and 5.23b from Section 5.3 of [9]. The equations in (86) and (87), are essentially equations 3.3 and 3.6 from [2], with the exception that the number of computations used for the FFT in 3.3 and 3.6 differ from that used for DFT in (86) and (87) and there is an *ML* multiplication in (87) versus just an L in equation 3.3 from [2]

For this analysis, *N* was set to 44000, *L* was set to 500, *K* was set to 8192. Each method was calculated for 2^3 to 2^8 number of Taus for decimation rates of D = 10, 100, 250, 400 and 500. The point in Figure 48 that the "Fine-Mode" Generic Frequency Domain method becomes more computationally efficient than the "Fine-Mode" Generic method is when the value of the curve becomes greater than one. It is clear from Figure 48 that the number of Taus plays a role in determining when one method is more computationally complex versus the other, thus confirming the hypothesis. For the test cases conducted, Figure 48 also indicates that for the number of Taus used for the "Fine-Mode" Generic Method (11), the "Fine-Mode" Generic method would always be more computationally efficient than the "Fine-Mode" Generic Frequency Domain method. This was the case for the testing conducted. If the number of used in the testing was > 32 then, the "Fine-Mode" Generic Frequency Domain method would have been more efficient.

For all of the tests, the Filter Bank method was clearly the best in accuracy. This is due to a lack of data reduction methods (e.g. decimation) in the algorithm. This method has significantly more total computations than any of the other methods (order of 10^6 more computations than the next computationally inefficient method). Unless there is no regard for computational complexity, this method should be avoided due to this large drawback.

For computational complexity, the "Fine-Mode" method ranks the best for all the tests. The "Fine-Mode" Generic, "Fine-Mode" Generic Frequency Domain, 2-D Cross Spectra, and Filter Bank methods in the order given, are how the remaining methods ranked in terms of computational complexity overall.

The "Fine-Mode" Generic and "Fine-Mode" Generic Frequency Domain methods had the same accuracy for all the tests. One possible reason for this result was because the same filter was used for both the methods. Even though the filter was applied in a different order for the methods, essentially, the multiplication of s_1, s_2, and the filter $h[n]$ was always occurring.

After the Filter Bank method, to quantify which methods were better for a particular frequency, bandwidth, and decimation, tables blah through blay were generated to highlight the best method(s). There was no "clear-cut" best overall method. However, most often for the high decimation rates, the "Fine-Mode" Generic and "Fine-Mode" Generic Frequency Domain methods had the best Tau and Doppler accuracy. For the low and medium decimation rates, the "Fine-Mode" method most often had the best Tau accuracy. The 2-D Cross Spectra method seemed to perform decently for the HF and VHF frequencies, but performed portly for the UHF frequency. The terminology used to represent the methods in the tables is as follows "Fine-Mode" method = FM, "Fine-Mode" Generic method = FMG, "Fine-Mode" Generic Frequency Domain method = FMGFD, and 2-D Cross Spectra method = 2DCS.

Chapter 5 Conclusion

From the testing, there is no best overall method that has the best accuracy and lowest computational complexity. For all the methods there is a trade-off in accuracy vs. computational complexity. For example, the Filter Bank method has the best accuracy at the cost of being one of the most computationally complex. The best method in terms of accuracy and complexity varies given a particularly frequency, sampling rate, and decimation rate.

The five methods examined have many advantages (data reduction techniques) and limitations. Some of these are summarized in Table 35.

Method	***Advantage***	***Disadvantage***
FB	• Best accuracy among all methods examined	• Not able to select number of Taus to use din computation • Not able to select number of Dopplers to be used in computation • Computationally complex
FM	• Most computationally efficient method • Able to select number of Taus to be used in computation	• Not able to select number of Dopplers to be used in computation • Limited by use of "all-ones" filter • Upper limit placed on L
FMG	• Ability to use any filter • Accuracy dependent upon filter used • Able to select number of Taus to be used in computation • Computationally efficient at low number of Taus • Ability to have $L > D$	• Not able to select number of Dopplers to be used in computation • Accuracy dependent upon filter used • Upper limit placed on L
FMGFD	• Ability to use any filter • Accuracy dependent upon filter used • Not able to select number of Dopplers to be used in computation • Computationally efficient at high number of Taus • Ability to have $L > D$	• Not able to select number of Taus to used in computation • Not able to select number of Dopplers to be used in computation • Accuracy dependent upon filter used • Computationally complex at low number of Taus • Upper and lower limit placed on L

Method	*Advantage*	*Disadvantage*
2DCS	• Ability to excise interference • Ability to use window of any shape that meets requirements • Accuarcy dependent upon win-dow(s) used • Ability to have $L > D$ •	• Computationally complex • Poor Tau accuracy • Not able to select number of Taus to used in computation • Not able to select number of Dopplers to be used in computation • Upper and lower limit placed on L

Table 35 Breakdown of Method Advantages and Disadvantages

Notes:
FB = Filter Bank
FM = Fine Mode
FMG = Fine Mode Generic
FMGFD = Fine Mode Generic Frequency Domain
2DCS = 2-D Cross Spectra

Some of the limitations are based on the design of the algorithm. The Filter Bank, "Fine-Mode" Generic Frequency Domain and 2-D Cross Spectra methods do not allow the user to select the number of Taus to use in computing the correlation surface. They provide an excessive amount of Taus. To limit this, for the Filter Bank method, the user of the algorithm would have to design a specific convolution/correlation routine that would select the Taus of interest. This may not be an easy task. An attempt can be made to limit the number of Taus for the "Fine-Mode" Generic Frequency Domain and 2-D Cross Spectra methods using extra care in picking the decimation (*D*), filter/window lengths (*L*), Doppler spacing, and/or the number of blocks (*M*) to provide DFT sizes that are as close to the desired number of Taus as possible. In some applications, this may not be possible. The user must be able to determine if he/she is able to live with the extra Taus and then attempt to try and limit the excess, before using these methods.

For Doppler the "Fine-Mode", "Fine-Mode" Generic, "Fine-Mode" Generic Frequency Domain and 2-D Cross Spectra methods provide an excess number of Dopplers. This also is due to the DFT sizes being $\geq$ the number of samples used for the DFTs being taken. Again, this can be attempted to be controlled by carefully choosing *D*, *L*, Doppler spacing, and *M*. As mentioned above, this may not be possible for some cases.

Another limitation observed is that D is limited by the sampling rate and the maximum expected Doppler frequency for all the methods that make use of data reduction techniques. This limits the amount of decimation that may be performed in certain scenarios, thus requiring more data for calculations than is desired. There is also a limit established on how large L may be. For the “Fine-Mode” and “Fine-Mode” Generic methods, only an upper bound exists. A lower bound and upper bound exist for the “Fine-Mode” Generic Frequency Domain and 2-D Cross Spectra methods. No limit exists for the Filter Bank method. Before using the “Fine-Mode” Generic Frequency Domain and 2-D Cross Spectra methods, it must be determined if the desired filter length, L, does not fall below the lower limit. In cases where a small L is desired, the “Fine-Mode” Generic Frequency Domain and 2-D Cross Spectra methods may not be suitable.

As noted in Chapter 4 Section 4.3, further study is needed to determine why the 2-D Cross Spectra method has poor Tau accuracy and to why the methods, with the exception of the Filter Bank method, did not perform as well as expected when comparing the results of the data with no perturbations to the Cramer-Rao lower bound. One path of examination that could yield a more accurate curve-fit result would be to use the results produced in this study for a coarse estimate. Then one could refine the coarse estimate by using a smaller subset of the data to along with the coarse estimate to produce a fine estimate for the emitter location as is proposed in [2]. Further testing using these methods should also be conducted to determine the results with perturbations. The tests conducted here did not include any interfering signals or added environmental noise because the purpose of the tests was to determine under certain circumstances the best method from a computational perspective with accuracy being a secondary benchmark. Accuracy of all the methods must be examined more thoroughly with modeling of real-life perturbations in order to correlate the accuracy with the computation complexity more realistically.

References

[1] S. Stein, "Algorithms for ambiguity function processing," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. ASSP-29, pp. 588 - 599, June 1981.

[2] R. Tolimieri and S. Winograd, "Computing the ambiguity function," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. ASSP-33, pp. 1239 - 1245, October 1985.

[3] L. Auslander and R. Tolimieri, "Computing decimated finite cross-ambiguity functions," *IEEE Trans. Acoust., Speech, and Signal Processing*, vol. ASSP-36, pp. 359 - 363, March 1988.

[4] L. Auslander, I. C. Gertner, and R. Tolimieri, "The discrete Zak transform application to time-frequency analysis and synthesis of nonstationary signals," *IEEE Trans. Signal Processing*, vol. 39, pp. 825 - 835, April 1991.

[5] G. A. Desjardins, "Frequency selective TDOA/FDOA cross-correlation," US Patent #5874916, held by Lockheed Martin Corp., Feb. 23, 1999.

[6] J. W. Cooley and S. Winograd, "A limited range discrete Fourier transform algorithm," *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, April 1980, pp. 213 – 217.

[7] L. H. Sibul and M. L. Fowler, "Optimum Array Processing In Inhomogeneous Random Fields," (Invited Paper), *Special Issue on Statistical Signal Processing*, *J. Instn. Electronics and Telecom. Engrs*., vol. 35, no. 2, pp. 98-104,1989.

[8] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform," *Proc. IEEE*, vol. 66, pp. 51 – 83, January 1978.

[9] P. Porat. A Course In Digital Signal Processing. John Wiley & Sons, Inc. New York. 1997.

[10] M. L. Fowler, "Correlation Processing," Lecture Notes. Binghamton University. http://www.ws.binghamton.edu/fowler/fowler%20personal%20page/EE521_files/IV-10%20Correlation%20Processing_2003.pdf.

[11] M. L. Fowler, "Cross-Spectrum-Based Interference Excision for TDOA/FDOA Estimation." Unpublished Notes. Binghamton University.

[12] "Data Bandplans." FCC: Industrial/Business Bandplans website. http://wireless.fcc.gov/services/ind&bus/data/bandplans.html.

Appendix A Matlab Code for Main Driver Script

```
% This script acts as the main driver to carry out a test case for all
% freqs, bandwidths, decimation rates
clear all;

% Set variables and arrays
tukey_win = 0;          % 1 = use Tukey window for FMGFD, 0 = fir1 filter
bw_to_run = 1;          % 1 = narrow, 2 = medium, 3 = wide
freq_to_run = 1;        % 1 = HF, 2 = VHF, 3 = UHF
method_to_run = 1;      % 1 = filter bank, 2 = FM, 3 = FMG, 4 = FMGFD, 5 = 2DCS
fhf = 5e6;              % Frequency of HF signal
fvhf = 50e6;            % Frequency of VHF signal
fuhf = 500e6;           % Frequency of UHF signal
f0 = [fhf fvhf fuhf];   % Vector of frequencies
c = 299792458;          % Speed of light constant
vel = 250;              % 250 m/s velocity for collection platforms
num_tau = 11;           % Number of time-delays

dec_hf = [6,12,700;12,24,1400;24,48,2801];
%dec_hf = [6,12,100;12,24,200;24,48,400];
dec_vhf = [6,12,83;12,24,166;24,48,333;];
dec_uhf = [6 7 8;12 14 18;24 28 33];
dopp_space = [0.2,0.4,0.5;0.2,0.4,0.5;0.2,0.4,0.5];
sampling_rate = [5.5e3,11e3,22e3;5.5e3,11e3,22e3;5.5e3,11e3,22e3];
dopp_freq = [3 16 -322];

% Set the state of the randn
randn('state',2);

% Set variables for test cases
freq = f0(freq_to_run);
fs = sampling_rate(freq_to_run,bw_to_run);
doppler_spacing = dopp_space(freq_to_run,bw_to_run);
doppler_freq = dopp_freq(freq_to_run);

% Create complex noise sequence
x = randn(1,11000) + j.*randn(1,11000);

% Determine the maximum doppler shift
max_doppler = ceil(freq*vel/c);

% Calculate the number of dopplers
num_dopplers = ceil(2*max_doppler/doppler_spacing);

% Create and apply filter to create s1 (pseudo-voice signal)
cut_off_freq = ((fs - .1*fs)/2)/(fs/2);
h = fir1(24, cut_off_freq);
s1 = filter(h,1,x);

% Apply Doppler shift to s2 signal
s2 = freq_shift(doppler_freq,0,s1,fs);

% Interpolate both streams to 4x
s1 = resample(s1,4,1);
s2 = resample(s2,4,1);

% Update fs
fs = fs * 4;
```

```
for ii = 1:3 % {
  % Determine decimation
  if (freq == fhf) % { HF freq
    D = dec_hf(bw_to_run,ii);
  elseif (freq == fvhf) % } VHF Freq {
    D = dec_vhf(bw_to_run,ii);
  else % } UHF Freq {
    D = dec_uhf(bw_to_run,ii);
  end % } if HF freq

  switch (method_to_run)
    case 1 % { Filter Bank - No decimation
      % Get Dopplers of interest
      if (freq == fhf) % {
        dopplers = -max_doppler:doppler_spacing:max_doppler;
        num_dopplers = length(dopplers);
      elseif (freq == fvhf) % } VHF Freq {
        dopplers = (doppler_freq-10):doppler_spacing:(doppler_freq+10);
        num_dopplers = length(dopplers);
      else % } UHF Freq {
        dopplers = (doppler_freq-10):doppler_spacing:(doppler_freq+10);
        num_dopplers = length(dopplers);
      end % } if HF Freq

      % Compute CAF
      [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fb(s1,s2,fs,num_tau,num_dopplers,doppler_spacing,dopplers);

      % Compute RMS error
      tau_error = sqrt(tau^2);
      doppler_error = sqrt((doppler)^2);
      fprintf(1,'TAU ERROR= %6.6f us\tDOPPLER ERROR= %f Hz\t# REAL ADDS = %.0f\t# REAL
MULTIPLIES = %.0f\n',...
          tau_error/1e-6,doppler_error,ceil(num_comp(1)),ceil(num_comp(2)));
      if (ii == 1) % {
        break; % Only need to one this case once for each bandwidth since no decima-
tion
      end %}
  case 2 % } Stein's Fine Mode {
      % Get number of samples
      N = length(s1);

      % Calculate filter length and decimation rate
      L = D;

      % Calculate the number of blocks
      M = ceil(abs((N-L))/D)+1;

      % Compute the cross-ambiguity function
      [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fmg(s1,s2,fs,ones(1,L),D,M,num_tau,doppler_spacing,max_doppler,0);

      % Compute RMS error
      tau_error = sqrt(tau^2);
      doppler_error = sqrt((doppler)^2);
      fprintf(1,'TAU ERROR= %6.6f us\tDOPPLER ERROR= %f Hz\t# REAL ADDS = %.0f\t# REAL
MULTIPLIES = %.0f\n',...
          tau_error/1e-6,doppler_error,ceil(num_comp(1)),ceil(num_comp(2)));
    case 3 % } FMG
      % Calculate filter cut-off frequency for filter
      cut_off_freq = max_doppler/(fs/2);

      % Get the number of samples
      N = length(s1);

      % Determine L
      L=D;

      % Create Generic filter
      h1 = fir1(L-1,cut_off_freq);

      % Calculate the number of blocks
      M = ceil((N-L)/D)+1;

      % Compute the cross-ambiguity function
      [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fmg(s1,s2,fs,h1,D,M,num_tau,doppler_spacing,max_doppler,1);

      % Compute RMS error
      tau_error = sqrt(tau^2);
      doppler_error = sqrt((doppler)^2);
      fprintf(1,'TAU ERROR= %6.6f us\tDOPPLER ERROR= %f Hz\t# REAL ADDS = %.0f\t# REAL
MULTIPLIES = %.0f\n',...
          tau_error/1e-6,doppler_error,ceil(num_comp(1)),ceil(num_comp(2)));
    case 4 % FMGFD
      % Get the number of samples
      N = length(s1);

      % Determine L
      L = D;

      % Get filter to use
      if (tukey_win == 0) % { fir1 filter
        cut_off_freq = max_doppler/(fs/2);
        win1 = fir1(L-1,cut_off_freq);
      else % } use Tukey Window {
        % Design filter for stream 1 - use Tukey window
        P = 0.80;
        win1 = tukeywin(L,P).';
      end % } if fir1 filter

      % Calculate the number of blocks
      M = ceil((N-L)/D)+1;

      % Compute the cross-ambiguity function
      [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fmgfd_old(s1,s2,fs,win1,D,M,num_tau,doppler_spacing,max_doppler);

      % Compute RMS error
      tau_error = sqrt(tau^2);
      doppler_error = sqrt(doppler^2);
      fprintf(1,'TAU ERROR= %6.6f us\tDOPPLER ERROR= %f Hz\t# REAL ADDS = %.0f\t# REAL
MULTIPLIES = %.0f\n',...
          tau_error/1e-6,doppler_error,ceil(num_comp(1)),ceil(num_comp(2)));
    case 5 % } 2DCS {
      % Get the number of samples
      N = length(s1);

      % Get length L.
      L = D;

      % Compute the cross-ambiguity function
      [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
twodcs(s1,s2,fs,L,D,num_tau,doppler_spacing,max_doppler);

      % Compute RMS error
      tau_error = sqrt(tau^2);
      doppler_error = sqrt((doppler_freq)^2);
      fprintf(1,'TAU ERROR= %6.6f us\tDOPPLER ERROR= %f Hz\t# REAL ADDS = %.0f\t# REAL
MULTIPLIES = %.0f\n',...
          tau_error,doppler_error,ceil(num_comp(1)),ceil(num_comp(2)));
    otherwise
      fprintf(1,'ERROR: UNDEFINED METHOD\n');
  end % } switch
end %} for ii = 1 all decimation rates for this sampling rate
```

Appendix B Matlab Code for freq_shift Function

```
function y = freq_shift(fnew,fold,x,fs);
% syntax: y = freq_shift(fnew,fold,x,fs);
%
% The function freq_shift takes the input data stream x and frequency
% shifts it by the amount (fnew-fold)/fs.  The shifted frequency stream y
% is returned
%
% INPUTS:
%   fnew - desired frequency to shift data in stream x to
%   fold - frequency that data in x is currently at
%   x    - complex signal data stream (1 X N) to frequency shift
%   fs   - sampling rate of signal stream x
%
% OUTPUTS:
%   y    - frequency shifted complex signal data stream
%
% ASSUMPTIONS:
%   This function assumes that all the inputs have been entered, thus no
%   checking is done on the number of inputs
%
% AUTHOR:
%   C. Yatrakis

% Calculate the difference in frequency between fnew and fold and normalize
% by fs
delta_f = (fnew-fold)./fs;

% Get the length of the signal stream x
N = length(x);

% Apply frequency shift
y = x.*exp(-j*2*pi*delta_f*(0:(N-1)));
```

Appendix C Matlab Code for curve_fit Function

```
function [tau,doppler] = curve_fit(A,tau_vec,dopp_vec);
% SYNTAX: [tau,doppler] = curve_fit(A,tau_vec,dopp_vec);
%
% DESCRIPTION:
% The function curve_fit takes a cross-ambiguity surface
% and applies a curve fit in the Tau direction and Doppler direction
% to obtain Tau and Doppler from the surface.  The Tau and Doppler
% are returned.
%
% INPUTS:
% A -  cross-ambiguity surface (Tau X Doppler)
% tau_vec - vector of time-delays used in computing surface (1xTau)
% dopp_vec - vector of Doppler frequenices used in computing surface
%            (1xDoppler)
%
% OUTPUTS:
% tau - Tau measured off the surface (scalar)
% doppler - Doppler measured off the surface (scalar)
%
% ASSUMPTIONS:
% There is enough room to the right and left of the surface peak
% to be able to pick points (e.g. surface peak not at edge)
%
% Interpolation to peak has already been taken care of
%
% AUTHOR:
% C. Yatrakis

% Get peak indices
[j1,dopp_pk_index] = max(max(A));
[j2,tau_pk_index] = max(max(A,[],2));

% Get Tau values to apply curve fit to
tau_values = A(tau_pk_index-2:tau_pk_index+2,dopp_pk_index);

% Get Tau curve fit coefficients
[tau_cf_coefs] = polyfit([tau_vec(tau_pk_index-2:tau_pk_index+2)],tau_values.',2);

% Get Doppler values to apply curve fit to
dopp_values = A(tau_pk_index,dopp_pk_index-2:dopp_pk_index+2);

% Get Doppler curve fit coefficients
[dopp_cf_coefs,s,mu] = polyfit([dopp_vec(dopp_pk_index-
2:dopp_pk_index+2)],dopp_values,2);

% Calculate Tau and Doppler from the surface peak
tau = -tau_cf_coefs(2)/(2*tau_cf_coefs(1));
doppler = -dopp_cf_coefs(2)/(2*dopp_cf_coefs(1));
```

Appendix D Matlab Code for fmg Function

```
function [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fmg(s1,s2,fs,h,D,M,T,doppler_spacing,max_doppler,fm_fmg_flag);
% SYNTAX: [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fmg(s1,s2,fs,h,D,M,T,doppler_spacing,max_doppler,fm_fmg_flag);
%
% DESCRIPTION:
% The function fmg performs the cross correlation of
% 2 signal streams using Seymour Stein's "Fine-Mode"
% method with a generic filter.  The implementation
% uses a polyphase filter approach.  The cross-ambiguity
% function is returned along with vectors of the time-delays
% and Dopplers for plotting. The time-delay and Doppler measured
% off the surface, and the number of computations it
% took to compute [computed using EQN 3.3 from Tolimieri &
% Winograds "Computing the Ambiguity Surface" paper are also returned
%
% INPUTS:
% s1 - complex signal data stream 1 (1xN)
% s2 - complex signal data stream 2 (1xN)
% fs - complex sampling rate of both signal streams in Hz (scalar)
% h - filter to be applied to the lag-product (1xL)
% D - decimation to be applied during correlation (scalar)
% dec - decimation to use during correlation (scalar)
% M - number of inner sums to apply in correlation(scalar)
% T - number of time-delays to use during correlation processing (scalar)
% doppler_spacing - Doppler spacing for correlation surface (scalar)
% max_doppler - maximum expected Doppler frequency (scalar)
% fm_fmg_flag - flag that tells if this is for fm or fmg mode (scalar).
%               Used for determining the number of computations
%
% OUTPUTS:
% A - complex cross-correlation function output (Tau X Dopplers)
% tau_vec - vector of taus used for plots (1 X Tau)
% dopp_vec - vector of Dopplers used for plots (1 X Doppler)
% tau - Tau measured off the surface (scalar)
% doppler - Doppler measured off the surface (scalar)
% num_comp - number of computations used to compute the cross-correlation
%           function (2x1) vector where the first row is real adds and
%           the second row is real multiplies
% LIMITATIONS:
% The size of both input streams must be the same
%
% AUTHOR:
% C. Yatrakis


% Get the number of samples in each stream
N = length(s1);

% Get the length of the filter
L = length(h);

% Zero pad s1 and s2 so they are divisible by L
num_zeros = L+D*(M-1)-N;
s1 = [s1 zeros(1,num_zeros)];

% Setup Tau range
if (mod(T,2) == 0) % { even
```

```
  min_tau = -T/2;
  max_tau = T/2 -1;
else % } odd {
  min_tau = -(T-1)/2;
  max_tau = (T-1)/2;
end % } if T is even

% Determine amount to zero-padding to get correct Doppler spacing
num_w_zp = ceil((fs/D)/doppler_spacing);

% Determine DFT size
dft_size = 2^(ceil(log2(num_w_zp)));

% Cross Correlate s1 and s2
for ii = min_tau:max_tau % { 0 to Num Taus - 1
  % Apply time-delay to s2
  if (ii >= 0) % {
    s2_star = conj([zeros(1,ii) s2(1,1:N-ii) zeros(1,num_zeros)]);
  else % } ii < 0 {
    s2_star = conj([s2(1,1-ii:N) zeros(1,-ii) zeros(1,num_zeros)]);
  end % } for ii >= 0

  % Form lag-product
  r_tau = s1.*s2_star;

  % Filter and decimate lag-product
  r_tau_tilda = upfirdn(r_tau,h,1,D);

  % Take DFT via FFT algorithm
  A(ii-min_tau+1,:) = fftshift(fft(r_tau_tilda,dft_size));

  % clear out filtered and decimated lag-product
  clear('r_tau_tilda','r_tau','s2_star');
end % } for ii = min_tau to max_tau

% Get Tau spacing
tau_spacing = inv(fs/D);

% Set tau_vec
tau_vec = [min_tau:max_tau]*tau_spacing;

% Calculate new doppler spacing for dopp_vec
fs_after_dec = fs/D;
dopp_space_after_dec = fs_after_dec/dft_size;

% Set dopp_vec
dopp_vec = [(-dft_size/2):((dft_size/2)-1)]*dopp_space_after_dec;

% Get tau and Doppler measurements from surface peak
[tau,doppler] = curve_fit(abs(A),tau_vec,dopp_vec);

% Get the number of computations needed for this method
if (fm_fmg_flag == 0) % { Stein's FM method
  real_mults = (T+1)*(4*N+2*dft_size*(log2(dft_size)-2)+4);
  real_adds = (T+1)*(2*N+2*M*2*L+3*dft_size*log2(dft_size)-2*dft_size+2);
  num_comp = [real_adds;real_mults];
else % } FMG method {
  real_mults = (T+1)*(4*N+4*M*(2*L+1)+2*dft_size*(log2(dft_size)-2)+4);
  real_adds = (T+1)*(2*N+2*M*(2*L+1)+3*dft_size*log2(dft_size)-2*dft_size+2);
  num_comp = [real_adds;real_mults];
end % } if FM method

return;
```

Appendix E Matlab Code for fmgfd Function

```
function [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fmgfd(s1,s2,fs,h,D,M,T,doppler_spacing,max_doppler);
% SYNTAX: [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fmgfd(s1,s2,fs,h,D,M,T,doppler_spacing,max_doppler);
%
% DESCRIPTION:
% The function fmgfd performs the cross correlation of
% 2 signal streams using Tolimieri and Winograd's "Fine-Mode"
% Generic Freqeuncy domain method.The cross-ambiguity
% function is returned along with vectors of the time-delays
% and Dopplers for plotting. The time-delay and Doppler measured
% off the surface, and the number of computations it
% took to compute [computed using EQN 3.2 from Tolimieri &
% Winograds "Computing the Ambiguity Surface" paper are also returned
%
% INPUTS:
% s1 - complex signal data stream 1 (1xN)
% s2 - complex signal data stream 2 (1xN)
% fs - complex sampling rate of both signal streams in Hz (scalar)
% h - window to be applied to the lag-product (1xL)
% D - decimation to be applied during correlation (scalar)
% dec - decimation to use during correlation (scalar)
% M - number of inner sums to apply in correlation(scalar)
% T - number of time-delays to use during correlation processing (scalar)
% doppler_spacing - Doppler spacing for correlation surface (scalar)
% max_doppler - maximum expected Doppler frequency (scalar)
%
% OUTPUTS:
% A - complex cross-correlation function output (Tau X Dopplers)
% tau_vec - vector of taus used for plots (1 X Tau)
% dopp_vec - vector of Dopplers used for plots (1 X Doppler)
% tau - Tau measured off the surface (scalar)
% doppler - Doppler measured off the surface (scalar)
% num_comp - number of computations used to compute the cross-correlation
%           function (2x1) vector where the first row is real adds and
%           the second row is real multiplies
%
% LIMITATIONS:
% The size of both input streams must be the same
%
% AUTHOR:
% C. Yatrakis

% Get the number of samples in each stream
N = length(s1);

% Get the length of the filter
L = length(h);

% Zero pad s1 and s2 so they are divisible by L
num_zeros = L+D*(M-1)-N; %
s1 = [s1 zeros(1,num_zeros)];
s2 = [s2 zeros(1,num_zeros)];

% Setup Tau range
if (mod(T,2) == 0) % { even
  min_tau = -T/2;
  max_tau = T/2 -1
else % } odd {
  min_tau = -(T-1)/2;
  max_tau = (T-1)/2;
end % } if T is even

% Determine DFT size
dft_size1 = 2^(ceil(log2(2*L-1)));

for mm = 0:M-1 % { 0 to Number of Blocks - 1
  % Get f_m and zero pad to length L+T
  f_mm= s1(mm*D+1:mm*D+L).*h;

  % Get s2_m, and zero pad both to 2L-1
  s2_mm = s2(mm*D+1:mm*D+L);

  % Take DFT of both streams
  F_mm = fftshift(fft(f_mm,dft_size1));
  S2_mm = fftshift(fft(s2_mm,dft_size1));

  % Multiply DFT of streams together
  W_mm = F_mm.*conj(S2_mm);

  % Take IDFT of both streams
  w_tilda(mm+1,:) = fftshift(ifft(W_mm,dft_size1));

  % Clear variables for next pass
  clear('W_m','f_mm','s2_mm','F_mm','S2_mm');
end % } for mm = 0 to Number of Blocks - 1

% Determine amount to zero pad columns of w_tilda_m to get correct Doppler
% spacing
num_w_zp = ceil((fs/D)/doppler_spacing);

% Determine DFT size
dft_size2 = 2^(ceil(log2(num_w_zp)));

% Get the number of Taus
[row,num_taus] = size(w_tilda);

% Compute correlation surface
A= fftshift(fft(w_tilda.',dft_size2,2),2);

% Calculate Tau spacing
tau_spacing = inv(fs/D);

% Set tau_vec
if (mod(num_taus,2) == 0) % { even
  tau_vec = [(-num_taus/2):(num_taus/2)-1].*tau_spacing;
else % } odd {
  tau_vec = [-((num_taus-1)/2):((num_taus-1)/2)].*tau_spacing;
end % } if num_taus is even

% Calculate new doppler spacing for dopp_vec
fs_after_dec = fs/D;
dopp_space_after_dec = fs_after_dec/dft_size2;

% Set dopp_vec
dopp_vec = [(-dft_size2/2):((dft_size2/2)-1)]*dopp_space_after_dec;

% Get tau and Doppler measurements from surface peak
[tau,doppler] = curve_fit(abs(A),tau_vec,dopp_vec);

% Get the number of computations needed for this method
% Get the number of computations needed for this method
real_mults = M*(4*L+6*dft_size1*(log2(dft_size1)-2)+12+4*dft_size1)+...
    dft_size1*(2*dft_size2*(log2(dft_size2)-2)+4);
real_adds = M*(2*L+9*dft_size1*log2(dft_size1)-6*dft_size1+6+2*dft_size1)+...
    dft_size1*(3*dft_size2*log2(dft_size2)-2*dft_size2+2);
num_comp = [real_adds;real_mults];

return;
```

Appendix F Matlab Code for fb Function

```
function [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fb(s1,s2,fs,T,num_dopplers,doppler_spacing,fdopp);
% SYNTAX: [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
fb(s1,s2,fs,T,num_dopplers,doppler_spacing,fdopp);
%
% DESCRIPTION:
% The function fb performs the cross correlation of
% 2 signal streams using the Filter Bank method.
% The implementation is a brute force approach.
% The cross-ambiguity function is returned along with
% vectors of the time-delays and Dopplers for plotting.
% The time-delay and Doppler measured off the surface,
% and the number of computations it took to compute [computed
% using EQN 2.3 from Tolimieri & Winograds "Computing the
% Ambiguity Surface" paper are also returned
%
% INPUTS:
% s1 - complex signal data stream 1 (1xN)
% s2 - complex signal data stream 2 (1xN)
% fs - complex sampling rate of both signal streams in Hz (scalar)
% T - number of time-delays to use during correlation processing (scalar)
% num_dopplers - number of doppler frequencies to use for correlation
%                surface (scalar)
% doppler_spacing - Doppler spacing for correlation surface (scalar)
% fdopp - Doppler frequencies of interest (1 x num_dopplers)
%
% OUTPUTS:
% A - complex cross-correlation function output (Num Taus X Num Dopplers)
% tau_vec - vector of time-delays used in computing correlation function
% dopp_vec - vector of frequencies used in computing the correlation
%            function
% tau - Tau measured off the surface (scalar)
% doppler - Doppler measured off the surface (scalar)
% num_comp - number of computations used to compute the cross-correlation
%           function (2x1) vector where the first row is real adds and
%           the second row is real multiplies
%
% LIMITATIONS:
% The size of both input streams must be the same
%
% AUTHOR:
% C. Yatrakis

% Get number of samples
N = length(s1);

% Get the tau spacing
tau_spacing = 1/fs;

for mm = 1:num_dopplers % { Dopplers of interest
  comp_exp = exp(-j.*2.*pi.*fdopp(mm).*[0:N-1]./fs);
  h_mm = s1.*comp_exp;

  % Compute the cross-ambiguity function
  A(:,mm) = xcorr(h_mm,s2).';
end % } for mm = Dopplers of interest

clear('s1','s2','h_mm','comp_exp');
```

```
% Get the number of taus used in computing the ambiguity function
[num_taus,col] = size(A);

% Set tau_vec
if (mod(num_taus,2) == 0) % { even
  tau_vec = [(-num_taus/2):(num_taus/2)-1].*tau_spacing;
else % } odd {
    tau_vec = [-((num_taus-1)/2):((num_taus-1)/2)].*tau_spacing;
end % } if num_taus is even

% Set dopp_vec
dopp_vec = [(-num_dopplers/2):((num_dopplers/2))].*doppler_spacing;

% Get tau and Doppler measurements from surface peak
[tau,doppler] = curve_fit(abs(A),tau_vec,dopp_vec);

% Get the number of computations needed for this method
real_mults = (num_dopplers+1)*(4*N + 4*(N+num_taus) + ...
        4*(N+num_taus)*(log2(N+num_taus)-2) +8) + ...
        2*(N+num_taus)*(log2(N+num_taus)-2) + 4;
real_adds = (num_dopplers+1)*(2*N + 2*(N+num_taus) + ...
    6*(N+num_taus)*log2(N+num_taus) - 4*(N+num_taus) + 4) + ...
    3*(N+num_taus)*log2(N+num_taus) - 2*(N+num_taus) + 2;
num_comp = [real_adds;real_mults];

return;
```

Appendix G Matlab Code for twodcs Function

```
function [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
twodcs(s1,s2,fs,L,D,T,doppler_spacing,max_doppler);
% SYNTAX: [A,tau_vec,dopp_vec,tau,doppler,num_comp] =
twodcs(s1,s2,fs,D,T,doppler_spacing,max_doppler);
%
% DESCRIPTION:
% The function fmgfd performs the cross correlation of
% 2 signal streams using Desjardin's 2-D Cross Spectra
% method.The cross-ambiguity function is returned along
% with vectors of the time-delays and Dopplers for plotting.
% The time-delay and Doppler measured off the surface, and the
% number of computations it took to compute are also returned
%
% INPUTS:
% s1 - complex signal data stream 1 (1xN)
% s2 - complex signal data stream 2 (1xN)
% fs - complex sampling rate of both signal streams in Hz (scalar)
% L - length of window for first stream (scalar)
% D - decimation to be applied during correlation (scalar)
% T - number of time-delays to use during correlation processing (scalar)
% doppler_spacing - Doppler spacing for correlation surface (scalar)
% max_doppler - maximum expected Doppler frequency (scalar)
%
% OUTPUTS:
% A - complex cross-correlation function output (Tau X Dopplers)
% tau_vec - vector of taus used for plots (1 X Tau)
% dopp_vec - vector of Dopplers used for plots (1 X Doppler)
% tau - Tau measured off the surface (scalar)
% doppler - Doppler measured off the surface (scalar)
% num_comp - number of computations used to compute the cross-correlation
%           function (2x1) vector where the first row is real adds and
%           the second row is real multiplies
%
% LIMITATIONS:
% The size of both input streams must be the same
%
% AUTHOR:
% C. Yatrakis

 % Get the number of samples
 N = length(s1);

 % Setup the tau range
 if (mod(T,2) == 0) % { even
   min_tau = -T/2;
   max_tau = T/2 -1;
 else % } odd {
   min_tau = -(T-1)/2;
   max_tau = (T-1)/2;
 end % } if T is even

% Create Tukey window for stream s2.  Use 80% Tukey window - arbitrary P
P = 0.80;
Lw2 = ceil((L+2*max_tau)/P);
win2 = tukeywin(Lw2,P).';

% Calculate the number of blocks
M = ceil((N-Lw2)/D)+1;
```

```
% Zero pad s1 and s2 so they are divisible by length Lw2
num_zeros = Lw2+D*(M-1)-N; %%% REVISIT THIS

s1 = [s1 zeros(1,num_zeros)];
s2 = [s2 zeros(1,num_zeros)];

% Design window for stream 1 - use another Tukey window
win1 = tukeywin(L,P).';

% zero pad win1 so it is length Lw2
num_zeros = Lw2-L;
win1 = [win1 zeros(1,num_zeros)];

% Compute DFT size
dft_size1 = 2^(ceil(log2(2*Lw2-1)));

for m = 0:M-1 % { 0 to Num Blocks - 1
  % Form the mth signal block
  s1mo_tilda = s1(m*D+1:m*D+Lw2);
  s2mo_tilda = s2(m*D+1:m*D+Lw2);

  % window both streams and zero pad to 2*Lw2 - 1
  S1mow1_tilda = fft(win1.*s1mo_tilda,dft_size1);
  S2mow2_tilda = fft(win2.*s2mo_tilda,dft_size1);

  % Form CS matrix and zero pad
  CS(m+1,:) = S1mow1_tilda.*conj(S2mow2_tilda);
  clear('s1mo_tilda','s2mo_tilda','S1mow1_tilda','S2mow2_tilda');
end % } for mm = 0 to Num Blocks - 1


% Determine the amount of zero padding to perform on columns of CS
% matrix to get correct Doppler spacing
num_w_zp_dopp = ceil((fs/D)/doppler_spacing);

% Determine the amount of zero padding to perform on rows of Adoppler
% matrix to get correct Tau spacing
num_w_zp_tau = max(dft_size1,T);

% Determine DFT size & compute CAF
dft_size_dopp = 2^(ceil(log2(num_w_zp_dopp)));
dft_size_tau = 2^(ceil(log2(num_w_zp_tau)));
Adoppler= fft(CS,dft_size_dopp,1);
clear('CS','s1','s2');
A = fftshift(ifft(Adoppler,dft_size_tau,2).');
clear('Adoppler');

% Set tau_vec
[num_taus,num_dopplers] = size(A);
tau_spacing = inv(fs/D);
if (mod(num_taus,2) == 0) % { even
  tau_vec = [(-num_taus/2):(num_taus/2)-1].*tau_spacing;
else % } odd {
  tau_vec = [-((num_taus-1)/2):((num_taus-1)/2)].*tau_spacing;
end % } if num_taus is even

% Calculate new doppler spacing for dopp_vec
fs_after_dec = fs/D;
dopp_space_after_dec = fs_after_dec/dft_size_dopp;

% Set dopp_vec
dopp_vec = [(-num_dopplers/2):((num_dopplers/2)-1)]*dopp_space_after_dec;
```

```
% Get tau and Doppler measurements from surface peak
[tau,doppler] = curve_fit(abs(A),tau_vec,dopp_vec);

% Get the number of computations needed for this method
real_mults = M*(8*Lw2+4*dft_size1*(log2(dft_size1)-2)+8+4*dft_size1)+...
    dft_size1*(2*dft_size_dopp*(log2(dft_size_dopp)-2)+4)+...
    dft_size_dopp*(2*dft_size_tau*(log2(dft_size_tau)-2)+4);
real_adds = M*(4*Lw2+6*dft_size1*log2(dft_size1)-2*dft_size1+4+2*dft_size1)+...
    dft_size1*(3*dft_size_dopp*log2(dft_size_dopp)-2*dft_size_dopp+2)+...
    dft_size_dopp*(3*dft_size_tau*log2(dft_size_tau)-2*dft_size_tau+2);
num_comp = [real_adds;real_mults];
return;
```