8-2021

Alternative Doppler Extraction for Indoor Communication Signals

Thomas L. Bradshaw
Utah State University

ALTERNATIVE DOPPLER EXTRACTION FOR INDOOR COMMUNICATION SIGNALS

by

Thomas L. Bradshaw

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Todd Moon, Ph.D. Major Professor	Jacob Gunther, Ph.D. Committee Member
Reyhan Baktur, Ph.D. Committee Member	D. Richard Cutler, Ph.D. Interim Vice Provost of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2021

ABSTRACT

Alternative Doppler Extraction for Indoor Communication Signals

by

Thomas L. Bradshaw, Master of Science

Utah State University, 2021

Major Professor: Todd Moon, Ph.D.
Department: Electrical and Computer Engineering

Passive bistatic radar has become a topic of recent research. This radar method uses opportunistic transmitters, such as radio broadcast towers and WiFi routers, as the transmitters in the bistatic radar system. Since opportunistic signals are widespread in society today, new opportunities for using these signals for passive bistatic radar are arising. Recent research has found ways to use these signals to track moving objects. WiFi signals are of particular interest for tracking people in a building for both surveillance and healthcare applications. WiFi signals are useful because most buildings have WiFi routers, and these signals are able to penetrate walls.

The common method of tracking a moving person using WiFi radar is the Doppler only method. This method extracts the Doppler frequencies from the Doppler-shifted echo received at a receiver. A Kalman filter or a particle filter can use the Doppler frequencies extracted at multiple receivers to track the moving person. Typically, the Doppler frequencies are extracted using the cross-ambiguity function (CAF). This function requires a reference signal without the Doppler echo in it, which means an extra antenna is needed to obtain this signal. This extra antenna requires more hardware and is not always feasible in application.

This thesis provides an alternative method to extract the Doppler frequencies on a communication signal. This alternative method removes the need for a reference antenna by using knowledge of the communication system and additional signal processing in order to separate the Doppler echo signal from the direct path signal. The method explored relies on the ability to remove the communication signal symbols from the received signal and accurately estimate the carrier frequency offset on the communication signal.

GNU Radio was used for implementing and testing this system. The system was tested with various devices and antenna configurations. Algorithms, such as the MUSIC algorithm, Viterbi algorithm, and BCJR algorithm, were applied to the signal in MATLAB in order to further improve the Doppler frequency extraction. In addition, various new techniques were developed to overcome limitations on the system.

(216 pages)

PUBLIC ABSTRACT

Alternative Doppler Extraction for Indoor Communication Signals

Thomas L. Bradshaw

Radar is a common detection system for detecting the speed and velocity of moving objects. Radar is performed by transmitting a radio signal in the direction of a moving object and then processing the received reflected radio waves. The radar system processes the received signal to extract the Doppler frequency of the reflected waves, which reveals information about the velocity of the object. In traditional radar, the system uses one antenna for both the transmitter and the receiver.

In passive bistatic radar, the transmit antenna and receiver antenna are separated. In addition, the radar system has no control over the transmitter. The transmitted signal is instead a signal of opportunity. Signals of opportunity include FM and AM radio signals, satellite GPS signals, and WiFi signals. The receiver in the radar system receives a signal of opportunity and extracts the Doppler frequency from the signal. The Doppler frequency can then be used to get the velocity of the object, which can be used for tracking the object.

In previous research, the methods of traditional radar were applied to passive bistatic radar using a WiFi signal in order to track a moving object within a building. Traditional radar methods are to compare the Doppler-shifted signal to the original signal. In passive radar, the original signal needs to be measured without being affected by the Doppler-shifted signal, which requires a second antenna to be placed by the transmitter.

The research presented suggests a new method of retrieving the Doppler frequencies from a communication signal. This new method does not require a separate antenna to get the original signal but uses knowledge about the communication signal to successfully measure the Doppler frequency on the signal. The system was implemented using GNU Radio, a software-defined radio library. Additional processing was done using MATLAB.

Dedicated to my wife and best friend, Kelsey.

ACKNOWLEDGMENTS

This thesis would not have come about without the support of many different people in my life. I am grateful for the education I have received because of them.

First, I would like to thank my wife, Kelsey, who comforted me through my many late nights of studying and supported me in my many endeavors over the past three years.

I would also like to thank my parents, who helped me love to learn and encouraged me to push beyond my comfort zone in order to accomplish more than I ever thought myself capable of.

I give a special thanks to my professor, Dr. Todd Moon, for sharing his knowledge with me. The countless hours he has spent teaching and mentoring me have shaped me into the engineer I am today.

I also thank my committee, Dr. Jacob Gunther and Dr. Reyhan Baktur, whose advice helped me push forward in my research.

I would like to give thanks to the countless friends and co-workers who also supported me through my research: Joseph Ipson, Mirelle DeSpain, Cayden Anderson, and Colton Lindstrom.

Finally, I would like to thank the Laboratory for Telecommunication Sciences for funding our research and helping me troubleshoot the many hardware problems I faced.

Thomas L. Bradshaw

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ADC	analog-to-digital converter
ARD	amplitude-range-Doppler
BCJR	Bahl, Cocke, Jelinek, and Raviv (developers of the algorithm)
BF	Balance Factor
CAF	cross-ambiguity function
DDS	direct digital synthesizer
DEM	Doppler extraction method
DFT	discrete Fourier transform
DTFT	discrete-time Fourier transform
FFT	fast Fourier transform
FM	frequency modulation
IQ	in-phase and quadrature
ISM	industrial, scientific, and medical
LUT	look-up table
MUSIC	multiple signal classification
PLL	phase-locked loop
QPSK	quadrature phase shift keying
RF	radio frequency
SIR	signal-to-interference ratio
SNR	signal-to-noise ratio
TED	timing error detector

CHAPTER 1
INTRODUCTION

Passive bistatic radar has become a topic of research interest in recent years. This method of radar relies on using opportunistic transmitters, such as radio broadcast towers and WiFi routers, as the transmitter in the bistatic radar system. These systems are advantageous over traditional bistatic radar systems as they reduce the hardware cost and can be used discreetly [1]. One particular area of research focuses on using WiFi signals to track human movement in a building. WiFi signals are of interest because of the widespread use of WiFi routers in both public and private buildings and the ability of the signals to penetrate walls [2].

In order to track human movement in a building, one method relies on collecting the Doppler echo signal at multiple receivers and then extracting the Doppler frequency path from this signal. This method collects the Doppler echo at each of the receivers and then inputs the Doppler frequency path into either a Kalman filter or a particle filter [3]. The Doppler frequency path is extracted through a matched filter in a cross-ambiguity function (CAF) [4]. The issue with this extraction method is the reliance on a reference signal in the matched filtering, which is needed to eliminate the direct path signal as it can easily overpower the Doppler echo. This reference signal is obtained by placing a reference antenna by the transmitter or using a narrow beam antenna pointed at the transmitter [5].

This thesis seeks to explore a new method of extracting the Doppler signal at each antenna without the need for a reference signal. This new method relies on demodulating the communication signal and removing the symbols in order to separate out the Doppler signal. This thesis describes the communication system built to test this new Doppler extraction method and additional processing for overcoming obstacles to the Doppler extraction.

1.1 Literature Review

1.1.1 Passive Bistatic Radar

Bistatic radar is a form of radar where the transmitter and receiver use separate antennas. This form of radar has been used since World War II, as recorded by Griffiths [1]. The particular advantage of separating the transmitting and receiving antennas is the greater protection provided to the system. Griffiths discusses the appeal of passive bistatic radar, where the transmitted signal is a signal of opportunity such as a communication signal (e.g. FM radio, WiFi). It is notable that at the time of Griffiths' article there were few passive radar systems implemented outside of research, as there were few situations where passive radar would work better than conventional radar.

Since the publication of Griffiths' paper in 2011, new applications for passive bistatic radar have arisen. FM-based passive radar has found new applications in tracking airborne targets, such as airplanes and drones [6–9]. WiFi-based passive radar has found applications in healthcare situations [4, 10] and surveillance [11].

1.1.2 FM-Based Passive Radar

Howland et al. looked at the feasibility of using FM signals over analog television signals in order to track aircraft. The method of obtaining Doppler information in a traditional radar system involves using the original signal as a matched filter for the Doppler echo signal to produce a range-Doppler matrix [6–8]. The waveform of analog television signals is less ideal for the matched filtering compared to the waveform of FM signals [6, pp. 1].

In passive bistatic radar, the original signal is replaced with a estimate of the original signal known as the reference signal. In order to separate the reference and echo signals, a few methods are employed. The reference signal is collected either by a dedicated receiver [8] or a receiver array with digital beamforming [6, 7]. The echo signal is obtained through an adaptive filter using the reference signal and a received signal at the antenna of interest [8]. Once the reference and echo signals are obtained, then the range-Doppler information is found using the amplitude-range-Doppler (ARD) surface calculated by [6, pp. 2]

$$|\Psi(\tau, f_d)| = \left| \sum_{n=0}^{N-1} s_{err}(n) s^*_{ref}(n-\tau) e^{j2\pi f_d n/N} \right|. \tag{1.1}$$

In this equation, s_{err} is the echo signal and s^*_{ref} is the conjugate of the reference signal. N is the number of bins on the frequency axis.

1.1.3 WiFi-Based Passive Radar

WiFi-based passive radar has gained recent interest because of the ability of WiFi signals to penetrate walls, which is useful for surveillance applications. A limiting factor in using WiFi is the overpowering strength of the direct path signal. The performance of the system is measured by the signal-to-interference ratio (SIR) rather than the signal-to-noise ratio (SNR) because of the direct path interference [2, pp. 1]. The CAF function computed by

$$CAF(\tau_d, f_d) = \sum_{n=0}^{N} r[n] s^*[n+\tau] e^{-j2\pi f_d n/N} \tag{1.2}$$

extracts Doppler information from a WiFi signal [12, pp. 2]. This equation is essentially the same as the ARD in (1.1), with $r[n]$ acting as the reference signal and $s[n]$ acting as the echo signal. The WiFi-based approach uses a similar method to the FM-based approach for measuring both of these signals. Typically, the reference signal is obtained by placing an antenna directly adjacent to the transmitter [3].

The CAF is adopted as the typical method in the literature for obtaining the range and Doppler information. Once the range and Doppler information is found, the Doppler information is used in an extended Kalman filter or a particle filter to track a moving person. Tan, Chetty, and Woodbridge used this method to track a person moving in a room [3] and to identify small human motions and hand gestures [4, 12]. Their work has been used and improved upon for various applications [10, 13, 14]. However, the method remains largely the same throughout the literature.

1.2 Background

The objective of the research was to develop a new method to extract the Doppler frequencies from a communication signal for the purpose of tracking a moving human in a building. This section reviews the background theory of the research.

1.2.1 Introduction to Digital Communications

Digital communication theory is described in detail in Dr. Rice's book [15]. This section gives a brief overview of signal space theory as it relates to digital communication theory. Chapter 2 goes into more of the digital communication theory as it relates to the Doppler signal extraction.

In digital communication theory, the communication signal is described using the concept of the signal space. The signal space is a vector space spanned by orthogonal waveforms. An example of a signal space is shown in Figure 1.1, which was the signal space used in this research. This specific signal space uses the orthogonal waveforms [15, pp. 223]

$$\begin{aligned} \psi_0(t) &= \sqrt{2}p(t)\cos(2\pi f_c t) \\ \psi_1(t) &= \sqrt{2}p(t)\sin(2\pi f_c t). \end{aligned} \tag{1.3}$$

In these equations, $p(t)$ is any unit energy pulse and f_c is the carrier frequency of the transmitted signal. The $p(t)$ chosen for this research was the square-root raised cosine pulse [15, pp. 637].

The points in the signal space each represent a number of bits and are known as symbols. The symbols in the signal space are collectively known as the signal constellation. For the signal space in Figure 1.1, each symbol represents two bits as shown. This constellation is known as the quadrature phase shift keying (QPSK) constellation.

The transmitted signal is formed by sending bits through the transmission system (see Section 2.3.1). In this system, the bits are mapped to symbols and then modulated by the waveforms of the signal space. For the QPSK constellation used in the research, each pair of bits is mapped to a constellation point denoted as (a_0, a_1). The signal created by taking

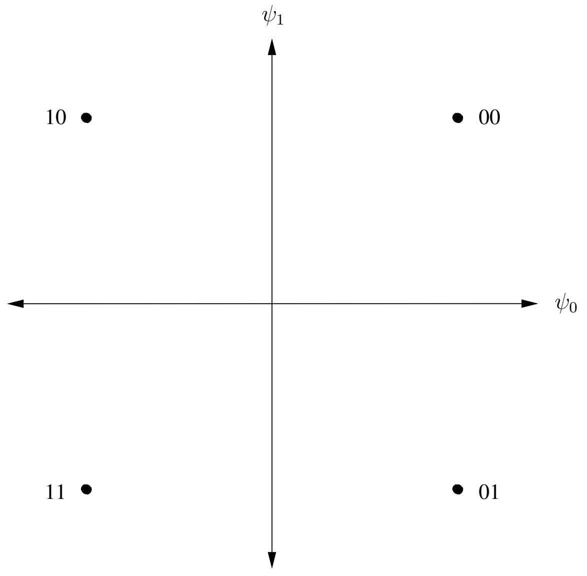


Fig. 1.1: QPSK constellation.

a series of these symbols is

$$s_m(t) = \sqrt{2}\Big(\sum_k a_0(k)p(t-kT_s)\cos(2\pi f_c t) - \sum_k a_1(k)p(t-kT_s)\sin(2\pi f_c t)\Big), \tag{1.4}$$

where k indexes the symbols. (1.4) can be written in a more convenient form of

$$s_m(t) = \sqrt{2}\big(I\cos(2\pi f_c t) - Q\sin(2\pi f_c t)\big). \tag{1.5}$$

In this equation, I is referred to as the in-phase portion of the signal and Q is referred to as the quadrature portion of the signal. Together, they are known as the in-phase and quadrature (IQ) signal.

The IQ signal can be represented as $I + jQ$, which in polar form is

$$s_b(t) = A(t)e^{j\phi(t)}, \tag{1.6}$$

where $I = A(t)\cos(\phi(t))$ and $Q = A(t)\sin(\phi(t))$. This form is convenient for representing the transmitted signal. Using this notation, (1.5) can be written as

$$s_m(t) = A(t)e^{j\phi(t)}e^{j2\pi fct} = A(t)e^{j(2\pi f_c t+\phi(t))}. \tag{1.7}$$

This is the transmitted signal in the bistatic radar system.

1.2.2 Bistatic Radar

Figure 1.2 shows a simple model for bistatic radar. There is a fixed transmitter at position $\mathbf{t} = (x_T, y_T)$, which transmits the modulated communication signal of (1.7). Let $\mathbf{q} = (x_R, y_R)$ be the position of a fixed receiver, $\mathbf{g}(t) = (x(t), y(t))$ be the position of a moving object at time t, and $\mathbf{v}(t) = (v_x(t), v_y(t))$ be the velocity vector for the moving object at time t.

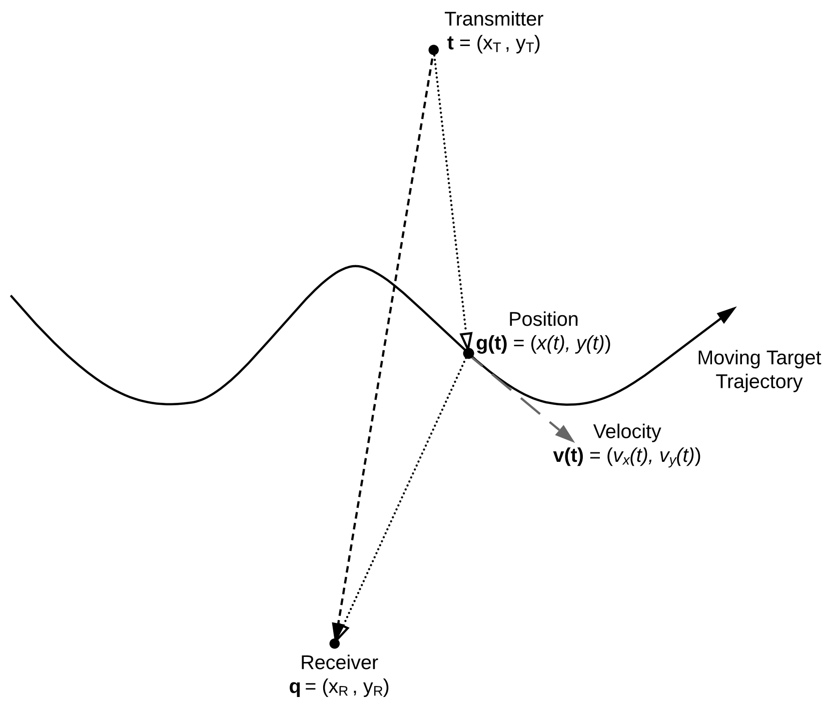


Fig. 1.2: Diagram of the direct and Doppler echo paths of a transmitted communication signal.

Now define $\hat{\mathbf{u}}_{\mathbf{t}}(t)$ as the unit vector pointing from $\mathbf{t}$ to $\mathbf{g}(t)$,

$$\hat{\mathbf{u}}_{\mathbf{t}}(t) = \frac{\mathbf{g}(t) - \mathbf{t}}{||\mathbf{g}(t) - \mathbf{t}||} \tag{1.8}$$

and define $\hat{\mathbf{u}}_{\mathbf{q}}(t)$ as the unit vector pointing from $\mathbf{q}$ to $\mathbf{g}(t)$,

$$\hat{\mathbf{u}}_{\mathbf{q}}(t) = \frac{\mathbf{g}(t) - \mathbf{q}}{||\mathbf{g}(t) - \mathbf{q}||}. \tag{1.9}$$

In both of these equations, $||\mathbf{x}|| = \sqrt{x^2 + y^2}$ is the euclidean distance of the vector $\mathbf{x}$. In addition, define $R_T(t)$ as the distance from the target to the transmitter and $R_R(t)$ as the distance from the target to the receiver.

The Doppler signal is found using the bistatic radar equation, [16, pp. 119]

$$f_d(t) = -\frac{1}{\lambda}\left(\frac{dR_T(t)}{dt} + \frac{dR_R(t)}{dt}\right) = -\frac{f_c}{c}\left(\frac{dR_T(t)}{dt} + \frac{dR_R(t)}{dt}\right), \tag{1.10}$$

where f_c is the carrier frequency of the transmitted signal, λ is the wavelength of the carrier frequency, and c is the speed of light. The Doppler signal is the scaled sum of the change in length of $R_T(t)$ and $R_R(t)$. The sign in (1.10) allows a decreasing distance to contribute positive Doppler.

For a general moving object and a moving transmitter or receiver (represented generically as $\mathbf{b}(t)$), the derivative of the path distance can be calculated as

$$\begin{aligned} \frac{dR}{dt} &= (\mathbf{v}_g(t) - \mathbf{v}_b(t)) \cdot \hat{\mathbf{u}}_{\mathbf{b}}(t) \\ &= (\mathbf{v}_g(t) - \mathbf{v}_b(t)) \cdot \frac{\mathbf{g}(t) - \mathbf{b}(t)}{||\mathbf{g}(t) - \mathbf{b}(t)||} \\ &= ((v_{gx}(t), v_{gy}(t)) - (v_{bx}(t), v_{by}(t))) \cdot \frac{(x(t), y(t)) - (x_b(t), y_b(t))}{||(x(t), y(t)) - (x_b, y_b)||}, \end{aligned} \tag{1.11}$$

which is derived in Appendix B.

In the human tracking situation, the transmitter and receiver are fixed. The Doppler signal in this situation is computed by applying (1.11) to the derivatives in (1.10) to get

$$\begin{aligned} \frac{dR_T(t)}{dt} &= \mathbf{v}(t) \cdot \hat{\mathbf{u}}_{\mathbf{t}}(t) = (v_x(t), v_y(t)) \cdot \frac{(x(t), y(t)) - (x_T, y_T)}{||(x(t), y(t)) - (x_T, y_T)||} \\ \frac{dR_R(t)}{dt} &= \mathbf{v}(t) \cdot \hat{\mathbf{u}}_{\mathbf{q}}(t) = (v_x(t), v_y(t)) \cdot \frac{(x(t), y(t)) - (x_R, y_R)}{||(x(t), y(t)) - (x_R, y_R)||}. \end{aligned} \tag{1.12}$$

Applying these derivatives to the bistatic radar equation shows that the Doppler signal is a function of the position and velocity of the moving object,

$$\begin{aligned} f_d\Big(x(t), y(t), v_x(t), v_y(t)\Big) = -\frac{f_o}{c}\Bigg(& (v_x(t), v_y(t)) \cdot \bigg(\frac{(x(t), y(t)) - (x_T, y_T)}{||(x(t), y(t)) - (x_T, y_T)||} \\ & + \frac{(x(t), y(t)) - (x_R, y_R)}{||(x(t), y(t)) - (x_R, y_R)||} \bigg) \Bigg). \end{aligned} \tag{1.13}$$

An extended Kalman filter can use multiple Doppler signals to estimate the position and velocity parameters in this equation. This results in the extended Kalman filter tracking a moving object. The objective is to extract the Doppler signals at multiple receivers for this Kalman filter.

When extracting the Doppler frequencies, it is useful to know the limits on the frequencies. The maximum Doppler frequency is determined by

$$|f_{max}| = \frac{2|V|f_c}{c}, \tag{1.14}$$

where V is the velocity of the moving object. For tracking a human, a reasonable maximum velocity is around 4 m/s, which is a little faster than the average running pace but corresponds to quick jerks of human movement. For this research, f_c was chosen to be around 905 MHz. This means the maximum Doppler could be as high as $|f_{max}| = 24$ Hz. Usually, people do not run in a building, but this value is reasonable for the test subject in Chapter 5.

1.3 Chapter Outlines

The following chapters describe an alternative method for extracting the Doppler signal. The system developed for testing the new Doppler extraction method is shown in Figure 1.3. This section gives a brief overview of each chapter. In this thesis, the Doppler echo signal refers to the signal received at the receiver; the Doppler signal or Doppler frequency path refers to the path of estimated Doppler frequencies of the Doppler signal.

Chapter 2 introduces the theory for extracting the Doppler signal from the communication system. Two methods of Doppler extraction are introduced. This chapter also goes over the implementation of the communication system, discussing both the hardware and software.

Chapters 3 and 4 discuss additional processing needed to extract the Doppler frequency path from the output of the receiver system. This processing is necessary because of the low SIR of the Doppler echo signal.

Chapter 3 describes the process of estimating the spectrogram of the receiver system output. The spectrogram is important for accurately measuring the frequency of the Doppler signal over time. The two main methods of spectrogram estimation discussed are the discrete Fourier transform (DFT) and the multiple signal classification (MUSIC) algorithm. In addition, a few other processing techniques are discussed.

Chapter 4 describes algorithms for estimating the Doppler frequency path from the spectrogram. There are various path estimation techniques used, such as the Viterbi and BCJR algorithms, which help pick the Doppler signal out of a signal with low SNR or SIR.

Chapter 5 discusses the results of various tests on the system. This chapter also compares the various devices used, the two Doppler extraction methods tested, and the improvements achieved using additional processing techniques.

Chapter 6 concludes the thesis, discussing how the objectives were met. It also describes future work on the subject.

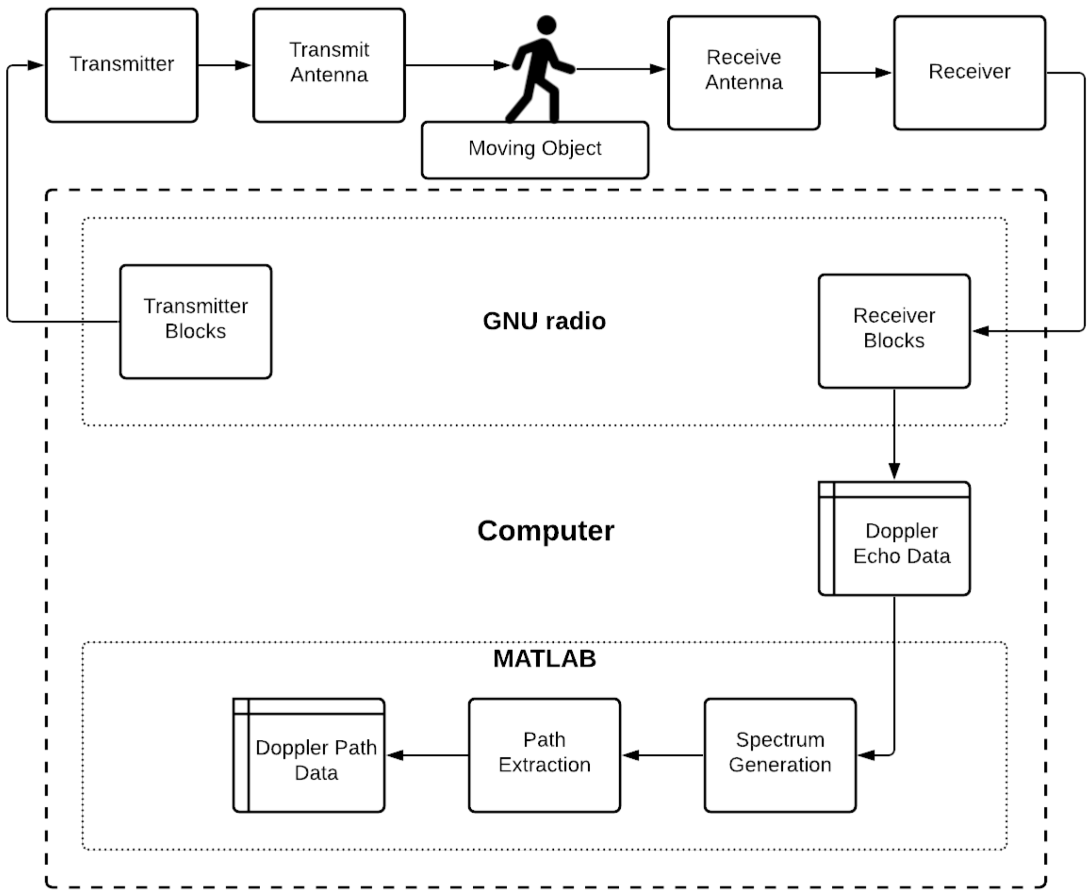


Fig. 1.3: Complete system high-level overview

CHAPTER 2
COMMUNICATION SYSTEM

This chapter looks at the theoretical and physical communication system used to extract the Doppler signal from the received communication signal. Section 2.1 gives an overview of the mathematical theory behind the Doppler extraction. Sections 2.2 and 2.3 review the hardware and software used to implement the theory.

2.1 Theory

The background of Section 1.2.2 introduced the transmitted signal, (1.7), in the bistatic radar system. This signal is transmitted and travels on many different paths to the receiver. The paths of interest, shown in Figure 1.2, are the direct path signal and the Doppler echo signal.

The receiver receives both the direct path signal and the Doppler echo along with additional clutter from other possible reflected paths. The theory ignores the additional clutter as it either is too weak or can be safely assumed to add to the direct path signal as the time delay between the clutter and the direct path is insignificant.

The direct path signal has a time delay of τ and an attenuation of α. The Doppler echo also has a time delay of $\tilde{\tau}$ and an attenuation factor of $\tilde{\alpha}$. The signal received at the receiver can be represented as a superposition of these two signals plus noise $w(t)$,

$$\begin{aligned} r(t) &= \alpha A(t-\tau)e^{j(2\pi f_c(t-\tau)(t-\tau)+\phi(t-\tau))} \\ &\quad + \tilde{\alpha}A(t-\tilde{\tau})e^{j(2\pi(f_c(t-\tilde{\tau})+f_d(t-\tilde{\tau}))(t-\tilde{\tau})+\phi(t-\tilde{\tau}))} + w(t-\tau). \end{aligned} \tag{2.1}$$

In this equation, $f_c(t)$ is the carrier frequency, which may change with time due to inconsistencies in the transmitter clock. The value $f_d(t)$ is the Doppler frequency. Substituting

$u = t - \tau$ in (2.1) leads to

$$\begin{aligned} r(t) &= \alpha A(u) e^{j(2\pi f_c(u)u + \phi(u))} \\ &\quad + \tilde{\alpha} A(u + \tau - \tilde{\tau}) e^{j(2\pi(f_c(u+\tau-\tilde{\tau}) + f_d(u+\tau-\tilde{\tau}))(u+\tau-\tilde{\tau}) + \phi(u+\tau-\tilde{\tau}))} + w(u). \end{aligned} \tag{2.2}$$

The value $\tau - \tilde{\tau}$ is the time difference between the signals. In the human tracking application, this time difference will be on the order of nanoseconds when $f_c = 905$ MHz. This time difference is insignificant compared to the length of the symbols. This time difference is also insignificant compared to how slowly the Doppler frequency and carrier frequency offset change. This leads to the approximation

$$r(t) \approx \alpha A(t) e^{j(2\pi f_c(t)(t-\tau) + \phi(t))} + \tilde{\alpha} A(t) e^{j(2\pi(f_c(t) + f_d(t))(t-\tilde{\tau}) + \phi(t))} + w(t). \tag{2.3}$$

The leftover τ and $\tilde{\tau}$ values multiplying $f_c(t)$ and $f_d(t)$ in the complex exponential terms generate phase terms, which can be absorbed into the constants α and $\tilde{\alpha}$,

$$\begin{aligned} r(t) &\approx \alpha e^{j2\pi f_c(t)\tau} A(t) e^{j(2\pi f_c(t)t + \phi(t))} \\ &\quad + \tilde{\alpha} e^{j2\pi(f_c(t)+f_d(t))\tilde{\tau}} A(t) e^{j(2\pi(f_c(t)+f_d(t))t + \phi(t))} + w(t) \\ &= \beta A(t) e^{j(2\pi f_c(t)t + \phi(t))} \\ &\quad + \tilde{\beta} A(t) e^{j(2\pi(f_c(t)+f_d(t))t + \phi(t))} + w(t). \end{aligned} \tag{2.4}$$

The received signal is sent through the receiver of Figure 2.1. This receiver takes in the received signal from (2.4), $r(t)$, and attempts to retrieve the symbols, a_k, from the signal.

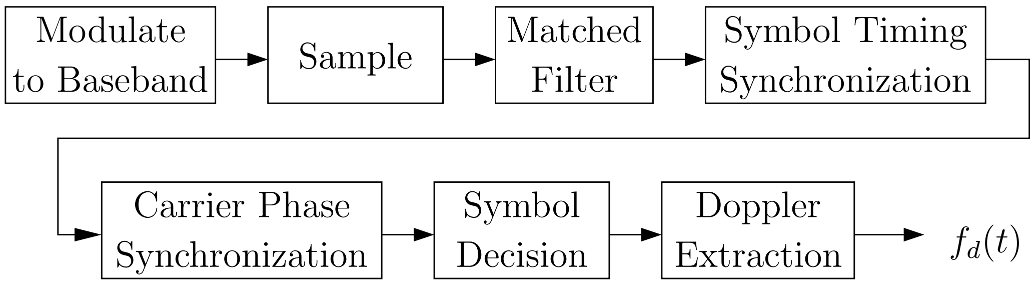


Fig. 2.1: Receiver high level overview.

In the process of retrieving the symbols, the Doppler signal can be extracted. Following the Doppler signal through this system reveals how to extract it.

The receiver first takes the received signal and attempts to demodulate it to baseband. This demodulation uses a set frequency, $\hat{f}_c(t)$, which is not equal to $f_c(t)$. The difference in these values comes from the clock errors on both the transmitter and receiver and from the phase shift of the path. This leads to the received signal not being completely demodulated to baseband,

$$\begin{aligned} r_b(t) &= r(t)e^{-j2\pi\hat{f}_c(t)t} \\ &= \left(\beta A(t)e^{j(2\pi f_c(t)t+\phi(t))} + \tilde{\beta}A(t)e^{j(2\pi(f_c(t)+f_d(t))t+\phi(t))} + w(t)\right)e^{-j2\pi\hat{f}_c(t)t} \\ &= \beta A(t)e^{j(2\pi f_e(t)t+\phi(t))} + \tilde{\beta}A(t)e^{j(2\pi(f_e(t)+f_d(t))t+\phi(t))} + w_b(t), \end{aligned} \tag{2.5}$$

where $f_e(t) = f_c(t) - \hat{f}_c(t)$ is the carrier frequency offset.

After being demodulated to baseband, an ADC samples the signal and sends it through a matched filter. The sampling is done at a sampling period of T,

$$\begin{aligned} r_b(nT) = \beta A(nT)&e^{j(2\pi f_e(nT)nT+\phi(nT))} \\ &+ \tilde{\beta}A(nT)e^{j(2\pi(f_e(nT)+f_d(nT))nT+\phi(nT))} + w_b(nT), \end{aligned} \tag{2.6}$$

and can be also written in discrete form as

$$r_b[n] = \beta A[n]e^{j(2\pi f_e[n]n+\phi[n])} + \tilde{\beta}A[n]e^{j(2\pi(f_e[n]+f_d[n])n+\phi[n])} + w_b[n]. \tag{2.7}$$

The signal is then sent through a matched filter. The matched filter has an impulse response of $h[n] = p[-n]$, with $p[n]$ as the pulse shaping function used by the transmitter. This pulse is known as it is defined by the communication protocol of the system. Sending

(2.7) through this filter results in

$$\begin{aligned}
y[n] &= r_b[n] * p[-n] \\
&= \big(\beta A[n] e^{j(2\pi f_e[n] n + \phi[n])}\big) * p[-n] + \big(\tilde{\beta} A[n] e^{j(2\pi(f_e[n]+f_d[n])n+\phi[n])}\big) * p[-n] \\
&\quad + w_b[n] * p[-n] \\
&= \big(\beta A[n] e^{j\phi[n]} e^{j2\pi f_e[n] n}\big) * p[-n] + \big(\tilde{\beta} A[n] e^{j\phi[n]} e^{j2\pi(f_e[n]+f_d[n])n}\big) * p[-n] \\
&\quad + v[n].
\end{aligned} \tag{2.8}$$

Due to the carrier frequency offset and Doppler frequency being relatively low frequencies, (2.8) can be approximated as

$$\begin{aligned}
y[n] \approx\ & \beta(A[n]e^{j\phi[n]} * p[-n])e^{j2\pi f_e[n]n} + \tilde{\beta}(A[n]e^{j\phi[n]} * p[-n])e^{j2\pi(f_e[n]+f_d[n])n} \\
& + v[n].
\end{aligned} \tag{2.9}$$

The symbol timing synchronization block next tries to sample $y[n]$ at the optimal times to retrieve the symbols. Assuming this block samples ideally, then the output of this block is [15, pp. 222]

$$\begin{aligned}
y[kT_s] &= \beta\big(A[kT_s]e^{j\phi[kT_s]} * p[-kT_s]\big)e^{j2\pi f_e[kT_s]kT_s} \\
&\quad + \tilde{\beta}\big(A[kT_s]e^{j\phi[kT_s]} * p[-kT_s]\big)e^{j2\pi(f_e[kT_s]+f_d[kT_s])kT_s} \\
&\quad + v[kT_s] \\
&= \beta\big(\sum_k a[k]p[k-kT_s] * p[-kT_s]\big)e^{j2\pi f_e[kT_s]kT_s} \\
&\quad + \tilde{\beta}\big(\sum_k a[k]p[k-kT_s] * p[-kT_s]\big)e^{j2\pi(f_e[kT_s]+f_d[kT_s])kT_s} \\
&\quad + v[kT_s] \\
&= \beta\frac{a[k]}{T}e^{j(2\pi f_e[kT_s]kT_s)} + \tilde{\beta}\frac{a[k]}{T}e^{j2\pi(f_e[kT_s]+f_d[kT_s])kT_s} + v[kT_s],
\end{aligned} \tag{2.10}$$

where $a(k)$ is the symbol at time k. The $1/T$ values can be absorbed into the scalar values and (2.10) can be written using the symbol indexing

$$\begin{aligned} y_k &= \beta a_k e^{j2\pi f_{e,k} k} + \tilde{\beta} a_k e^{j2\pi (f_{e,k} + f_{d,k})k} + v_k \\ &= \left(\beta a_k + \tilde{\beta} a_k e^{j2\pi f_{d,k} k} \right) e^{j2\pi f_{e,k} k} + v_k. \end{aligned} \tag{2.11}$$

The next block, the carrier phase synchronization block, uses a phase-locked loop (PLL) to remove the carrier frequency offset value. Ideally, the output of this block is

$$z_k = \beta a_k + \tilde{\beta} a_k e^{j2\pi f_{d,k} k} + \nu_k. \tag{2.12}$$

The final block of the receiver, the symbol decision block, takes the output of the carrier phase synchronization block and makes a decision on the symbols. The output of this block is

$$x_k = \hat{a}_k. \tag{2.13}$$

When making the decision, the Doppler frequency and any additional noise are all lost.

The Doppler signal is extracted by using the output of the carrier phase synchronization block, z_k, and the output of the symbol decision block, x_k,

$$\begin{aligned} \mathrm{DEM}_1(k) \triangleq \frac{z_k - x_k}{x_k} &= \frac{\beta a_k + \tilde{\beta} a_k e^{j2\pi f_{d,k} k} + \nu_k - \hat{a}_k}{\hat{a}_k} \\ &= \beta - 1 + \tilde{\beta} e^{j2\pi f_{d,k} k} + \text{noise}. \end{aligned} \tag{2.14}$$

This equation assumes the symbol estimation is accurate in order for $a_k = \hat{a}_k$. (2.14) is referred to as the First Doppler Extraction Method (DEM).

Once the data is manipulated in the form of (2.14), additional processing removes the noise and the DC components of this signal and then estimates the Doppler signal, as discussed in Chapters 3 and 4.

2.1.1 Second Doppler Extraction Method

The Doppler extraction theory presented assumes the carrier phase synchronization block ideally removes the carrier frequency offset and outputs the symbols in (2.12). However, this assumption is incorrect and the true output is closer to

$$z_k = \beta a_k + K a_k e^{j2\pi f_{d,k} k} + \tilde{K} a_k e^{j2\pi f_{d,k} k} + \nu_k. \tag{2.15}$$

In this equation, K and $\tilde{K}$ are unknown magnitudes, which vary with the frequency of the Doppler. Their exact values are hard to calculate due to the nonlinear nature of the PLL. A further discussion on how the Doppler signal affects the communication signal and on how the PLL operates sheds insight into why (2.15) is a closer representation of the output.

Figure 2.2 shows how the Doppler affects the symbols in the symbol space. A received symbol a_k has a small rotating Doppler offset added to it, as seen by rewriting (2.12) as

$$z_k = a_k(\beta + \tilde{\beta} e^{j2\pi f_{d,k} k}) + \nu_k. \tag{2.16}$$

This offset rotates the Doppler around the transmitted symbol. When there is a carrier

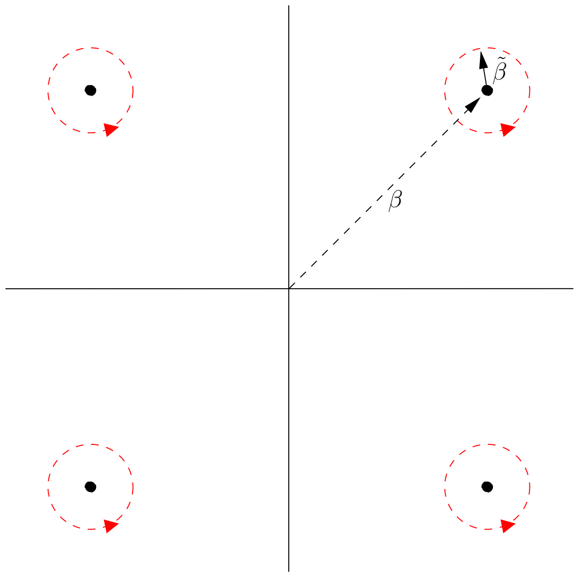


Fig. 2.2: Doppler rotations around symbols in the signal space.

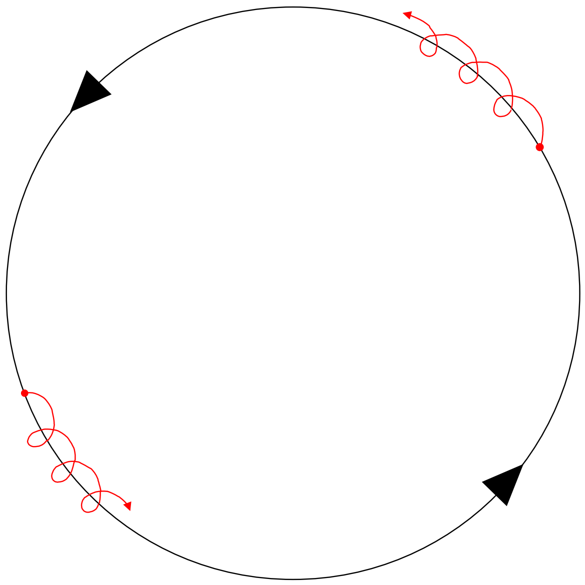


Fig. 2.3: Doppler rotations around symbols in the signal space with a rotating symbol constellation.

frequency offset, this whole constellation is rotating. In this situation, the Doppler rotates as helices, which is seen in Figure 2.3.

The carrier phase PLL, shown in Figure 2.4, attempts to correct for the rotations caused by the carrier frequency offset. The PLL operates by taking in one of the sampled symbols, y_k, and multiples it by $e^{-j\hat{\theta}}$. The value $\hat{\theta}$ is an estimate of how much the carrier frequency offset has rotated the symbol in the symbol space from the correct value, which is assumed to be the closest symbol. The multiplication result becomes the output z_k.

In order to calculate the θ for the next iteration, z_k is sent through a symbol decision block to estimate which of the symbols z_k might be. Then the error between the estimated symbol and the actual symbol is calculated. This error accounts for errors from the carrier phase and frequency offsets, and from the Doppler frequency as both values pull the symbol away from the decision points. This means the error is a combination of both values.

This error is sent into the loop filter. The loop filter's purpose is to track the frequency and phase of the carrier frequency offset when it is locked on, which can take a few iterations of the loop to do. It has two adjustable parameters, K_1 and K_2, which control the bandwidth of the filter. If the carrier frequency offset is less than the Doppler frequency shift, then the

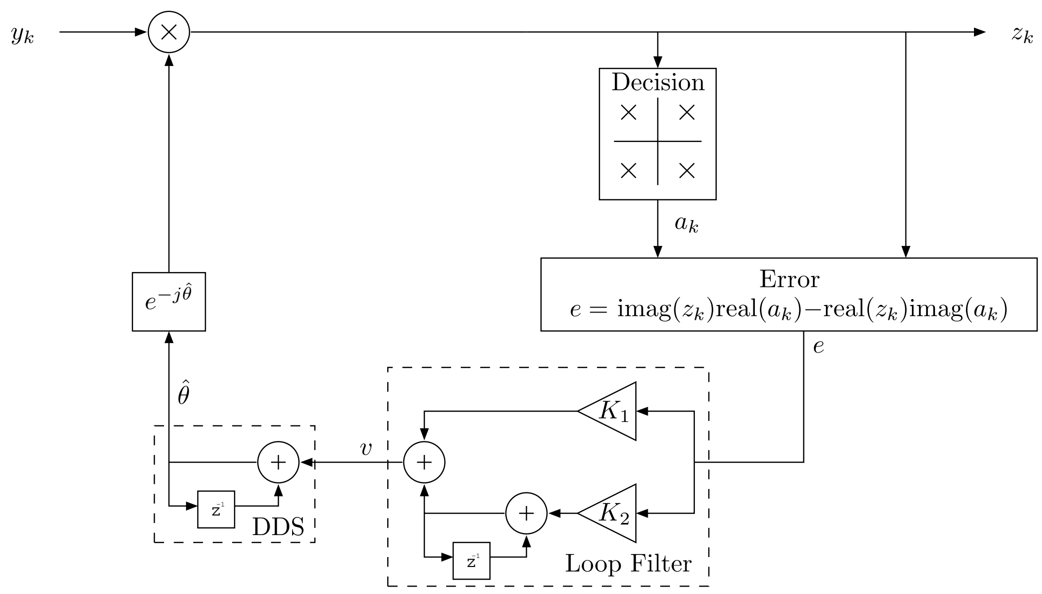


Fig. 2.4: Flowchart of the carrier phase PLL.

filter bandwidth can be set to filter out the Doppler frequency. However, the more realistic situation is the carrier frequency offset is much larger than the Doppler frequency shift and the loop filter tracks both frequencies.

The output of the loop filter is sent through a direct digital synthesizer (DDS) to update the estimate on θ. This means both the carrier frequency offset and the Doppler frequency influences θ. This leads to two Doppler peaks at both the positive and negative Doppler frequencies. It is ambiguous which frequency is real and which was created by the PLL. This becomes a major problem as the sign of the Doppler is crucial in determining if an object is moving towards or away from the receiver. Therefore, another method of extracting the Doppler frequency was developed to avoid this ambiguity.

This new method circumnavigates the PLL by using the value y_k instead of z_k in the Doppler extraction. This value is the output of the symbol timing synchronization block given by (2.11). Taking this value and the estimated symbols, a_k, the Doppler can be extracted using

$$\begin{aligned}\mathrm{DEM}_2(k) \triangleq \frac{y_k}{x_k} &= \frac{\left(\beta a_k + \tilde{\beta} a_k e^{j2\pi f_{d,k} k}\right) e^{j2\pi f_{e,k} k} + \nu_k}{\hat{a}_k} \\ &= \beta e^{j2\pi f_{e,k} k} + \tilde{\beta} e^{j2\pi (f_{e,k} + f_{d,k}) k} + \text{noise}.\end{aligned} \tag{2.17}$$

This equation is referred to as the Second DEM.

The Fourier transform of this value is two Dirac deltas, one at the carrier frequency offset and the other at the carrier frequency offset plus the Doppler shift. The difference between these values is the Doppler frequency. Therefore, this Second DEM is another way to extract the Doppler signal and does not suffer from the problems in the PLL. Chapters 3 and 4 go into more detail about how the Doppler frequency path can be extracted from this second method.

2.2 Hardware Description

The hardware in the implemented system consists of a transmitter, one or more receivers, antennas, one or two amplifiers, and one or more computers. The multiple receivers, amplifiers, and computers depend on the test setup. Figure 2.5 shows the hardware for the simplest setup. This section goes over all of the hardware used, but the specific configuration for each test is listed in Chapter 5.

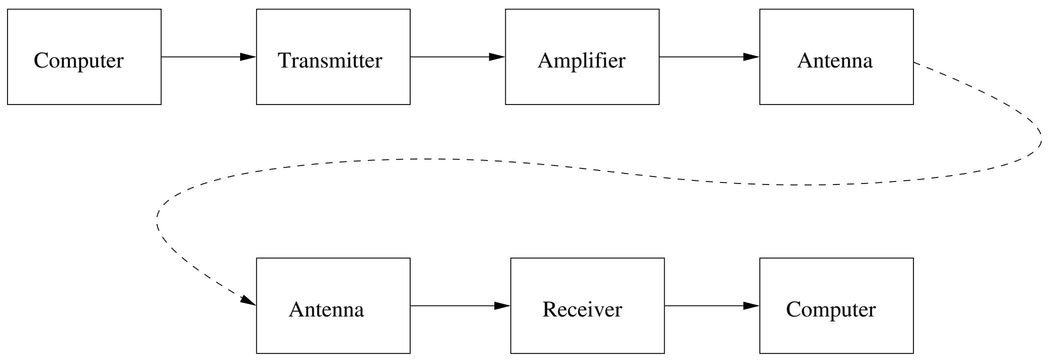


Fig. 2.5: Hardware overview.

2.2.1 Transmitters and Receivers

The transmitter and receiver devices in Table 2.1 were used in various tests. All of these devices except the RTL-SDR are made by Ettus Research. These devices were chosen because GNU Radio has built-in support for them.

The RTL-SDR dongle strictly operates as a receiver and cannot be configured as a transmitter. The RTL-SDR device was used in some of the early tests but suffered the most from clock drift and noise, making it harder to extract the Doppler frequency path. The advantage of the RTL-SDR receiver is its low cost and small package.

The Ettus Research devices are listed in increasing price in Table 2.1. The B200 and B210 devices are very similar in their hardware, differing mostly in the number of ports each device has. The B200mini is a miniature version of the B200 device but lacks a GPSDO clock. This hindered the usefulness of the B200mini as the GPSDO clocks are significantly better than the onboard clocks.

The E320s are more advanced devices capable of running software on the devices themselves. These devices are more expensive than the previous devices and would have had better clocks, except there is a major problem with the devices. The E320s introduce extra

Table 2.1: Transmitter and receiver devices.

Device	Frequency Range	Max Sample Rate	Ports	Clocks
RTL-SDR	24 - 1766 MHz	3.2 MS/s	1 Rx	On-board
B200	70 - 6000 MHz	61.44 MS/s	1 Tx, 1 Rx	On-board, GPSDO, and reference
B200mini	70 - 6000 MHz	61.44 MS/s	1 Tx, 1 Rx	On-board and reference
B210	70 - 6000 MHz	61.44 MS/s	2 Tx, 2 Rx	On-board, GPSDO, and reference
E320	70 - 6000 MHz	61.44 MS/s	2 Tx, 2 Rx	On-board, GPSDO, and reference
N310	10 - 6000 MHz	153.6 MS/s	4 Tx, 4 Rx	On-board, GPSDO, and reference

phase jitter on their clocks. The exact cause of this phase jitter was never determined. However, the E320s were the only devices with an adjustable master clock. It is theorized that the clock on the devices would oscillate between two frequencies in an attempt to get as close to the desired clock value as possible, but this oscillation would yield extra phase jitter on the clock. This jitter shows up as extra noise in the same frequency band as the Doppler and makes it harder to extract the Doppler from the communication signal. Therefore, the E320 devices were used sparingly in testing.

The final Ettus Research devices, the N310s, were obtained late in the testing process. These devices are the most expensive but have the best clocks. These devices also have more ports than the other devices. The N310s have four ports for transmitting or receiving. This was useful for many of the later tests where multiple receivers were used in different positions.

2.2.2 Antennas

Two different types of antennas were used for this system. Table 2.2 shows a few specifications for these antennas. The Heyrtz antenna was used for the earliest tests. Due to the transmit frequency of the tests being in the 902 – 928 frequency range, this antenna attenuated the signal significantly, making it useless when the transmitter and receivers were spread out further than a few meters. The HyperLink antennas were obtained when more transmitter power was needed. These antennas allowed the signal to travel further than a couple of meters.

Table 2.2: Antenna parameters.

Antenna	Frequency Range	Gain	Size
Heyrtz Retractable Antenna	144 - 430 MHz	2.15 dBi	420 mm extended, 140 mm retracted
HyperLink Wireless "Rubber Duck" Antenna	902 - 928 MHz	5 dBi	375 mm

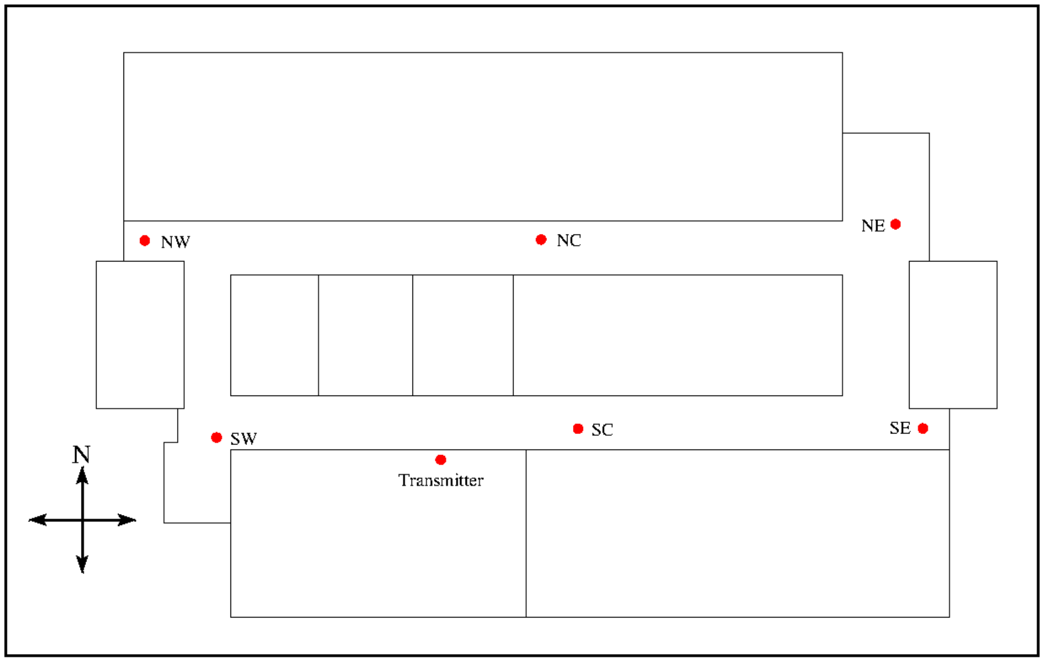


Fig. 2.6: Building antenna positions.

The placement of the antennas on both the transmit and receive side of the communication system varied from test to test. Initially, the antennas were placed side by side to test how much Doppler signal was reflected directly back.

The next tests used antennas in the ceiling throughout the entire second floor of the Sant Engineering Innovation building at Utah State University. The placement of the ceiling antennas is shown in Figure 2.6. The transmit antenna, also shown in this figure, was mounted on a tripod and moved around to different positions.

2.2.3 Other Hardware

The other hardware used were a couple of amplifiers and a couple of computers for sending and receiving the data.

The amplifiers used were Shireen 900 MHz 2 Watt amplifiers. These devices were used on the transmitter in order to increase the power transmitted. This was needed to make sure there was enough power for the receivers to pick up the Doppler echos.

The computers used were either a Lambda Labs workstation or Lambda Labs laptops, although other desktop or laptop computers could also work. The computers all ran Ubuntu 18.04.5 and were installed with GNU Radio 3.8 and all the required packages to work with the transmitter and receiver devices mentioned in Section 2.2.1.

2.3 GNU Radio QPSK System

The communication protocol chosen for this research was QPSK, which was introduced in Section 1.2.1. QPSK was chosen for the prototype system rather than the WiFi communication protocol as it was a simpler system to test out the basic theory on. Each data symbol in QPSK is one of four possible constellation points transmitted on an IQ signal, with each point encoding two bits of binary. The IQ components are transmitted by modulating the in-phase component with a cosine wave and the quadrature component with a sine wave before adding them together and transmitting them. This allows the two components to be separated at the receiver by a similar demodulation. More information about this standard and basic communication theory can be found in Dr. Rice's book on digital communications [15].

GNU Radio is a software-defined radio library used for easy prototyping and implementation of communication systems. It was chosen as the software used to implement this system because the communication system can be implemented in predefined blocks and easily modified for the desired application. The following sections discuss how this system was implemented in GNU Radio. The parameters listed in this chapter depend on the test being run. The specifics are given in Chapter 5.

2.3.1 Transmitter

The transmitter portion of the QPSK system is easy to implement in GNU Radio. Figure 2.7 shows the operations in the transmitter, and Figure 2.8 shows the two GNU Radio blocks containing all of these operations.

The Constellation Modulator block, shown on the left of Figure 2.8, accomplishes the serial-to-parallel conversion, look-up tables (LUT), and pulse shaping filter operations of

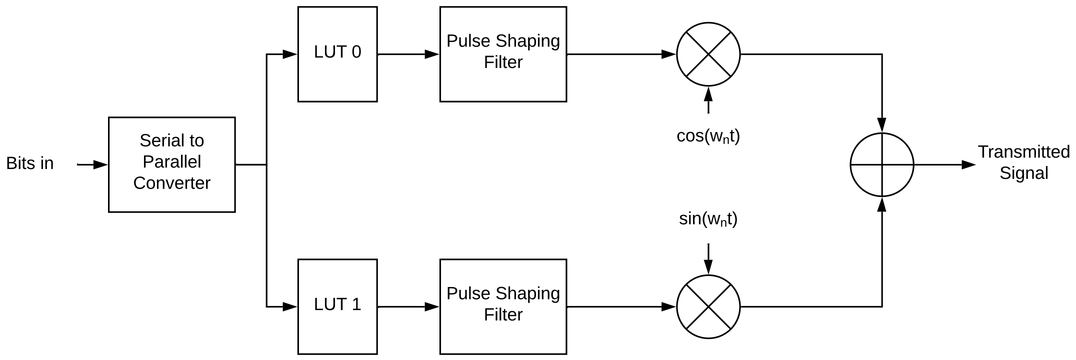
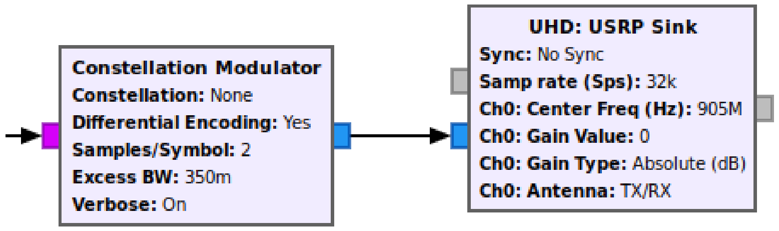


Fig. 2.7: Transmitter system high level overview.



Constellation Modulator
Constellation: None
Differential Encoding: Yes
Samples/Symbol: 2
Excess BW: 350m
Verbose: On

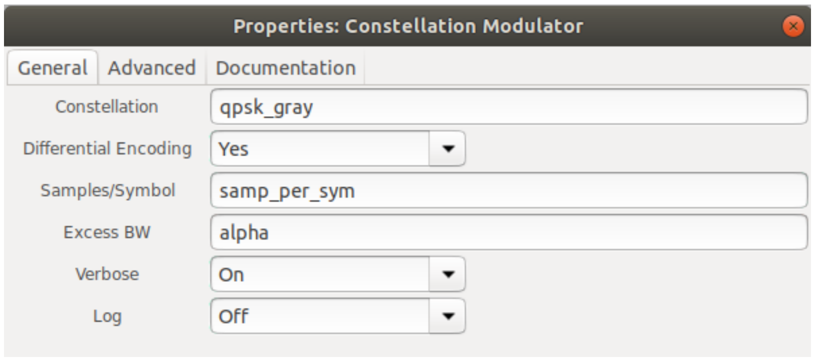
UHD: USRP Sink
Sync: No Sync
Samp rate (Sps): 32k
Ch0: Center Freq (Hz): 905M
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: TX/RX

Fig. 2.8: Transmitter system GNU Radio flowgraph.

the transmitter system. The input stream requires a stream of bytes. The output stream is the IQ data in a complex data format $(I + jQ)$.

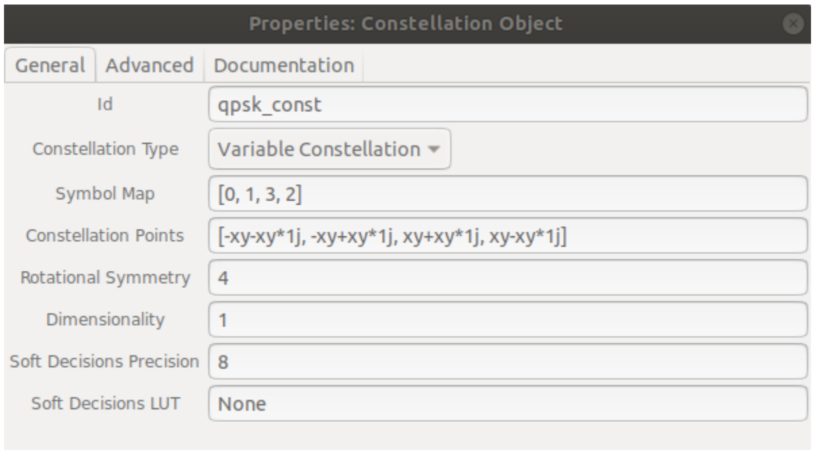
The Constellation Modulator block gives a little control over this process, as seen in Figure 2.9. For the implemented system, the constellation was set as a QPSK constellation using a Constellation Object block (Figure 2.10). The system also used differential encoding, although this parameter did not affect the ability to extract the Doppler signal. The samples per symbol and the excess bandwidth were set as variables as they influenced the performance of the system.

The second block in Figure 2.8 is the UHD: USRP Sink. This block takes care of modulating the IQ data, adding the IQ components together, and transmitting the resulting signal. This block is specifically for the Ettus Research devices mentioned above. Other transmitter devices may have a similar block.



Constellation	qpsk_gray
Differential Encoding	Yes
Samples/Symbol	samp_per_sym
Excess BW	alpha
Verbose	On
Log	Off

Fig. 2.9: Constellation Modulator properties.



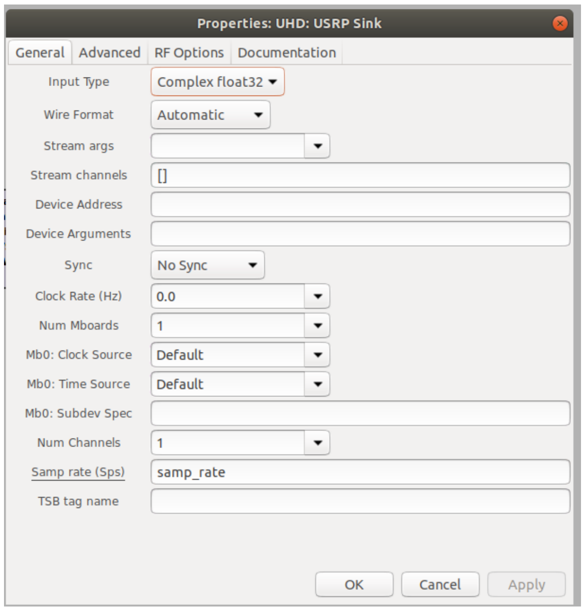
Id	qpsk_const
Constellation Type	Variable Constellation
Symbol Map	[0, 1, 3, 2]
Constellation Points	[-xy-xy*1j, -xy+xy*1j, xy+xy*1j, xy-xy*1j]
Rotational Symmetry	4
Dimensionality	1
Soft Decisions Precision	8
Soft Decisions LUT	None

Fig. 2.10: Constellation Object properties.

The goal of the transmitter blocks is to communicate with a transmitter device. The modulation and transmission of the data occur within the device. This block sends the data and any important parameters needed for setting up the transmission. The parameters can be controlled using the properties of the appropriate transmitter block. The important UHD: USRP Sink block properties are shown in Figures 2.11 and 2.12.

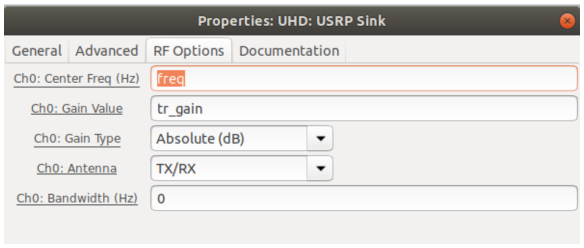
This first page of parameters in Figure 2.11 allows for control of the initialization parameters, such as the clock rate, sample rate, and source of the clock. These values all depended on the specific device being used.

The third page of parameters in Figure 2.12 allows for control of the gain and center frequency of the transmission. The gain was allowed to change on run-time from 0 - 90 dB. The center frequency chosen for the tests ranged in the ISM band of 902 - 928 MHz. In



Input Type	Complex float32
Wire Format	Automatic
Stream args	
Stream channels	[]
Device Address	
Device Arguments	
Sync	No Sync
Clock Rate (Hz)	0.0
Num Mboards	1
Mb0: Clock Source	Default
Mb0: Time Source	Default
Mb0: Subdev Spec	
Num Channels	1
Samp rate (Sps)	samp_rate
TSB tag name	

Fig. 2.11: UHD:USRP Sink general properties.



Ch0: Center Freq (Hz)	freq
Ch0: Gain Value	tr_gain
Ch0: Gain Type	Absolute (dB)
Ch0: Antenna	TX/RX
Ch0: Bandwidth (Hz)	0

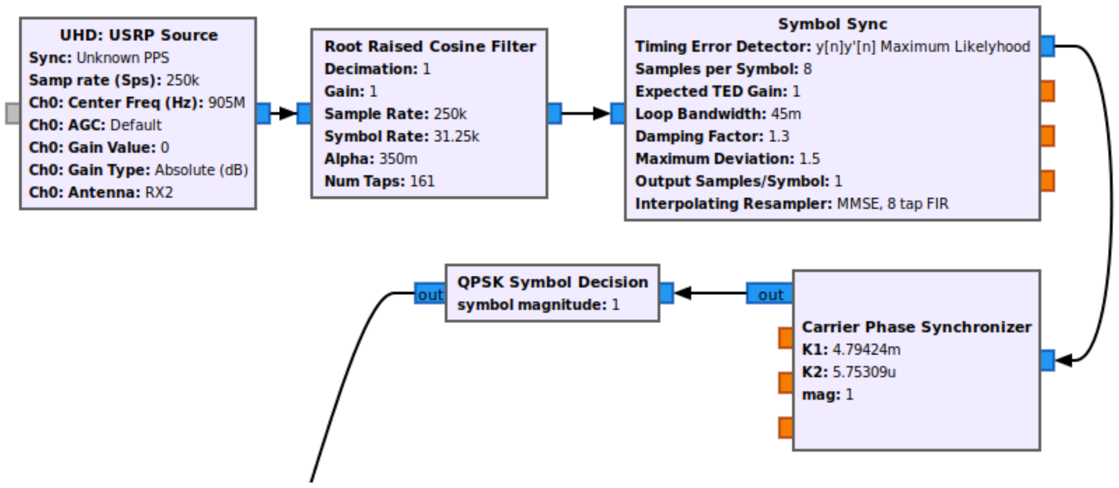
Fig. 2.12: UHD:USRP Sink RF properties.

addition, the desired transmit port for the device needs to be specified.

2.3.2 Receiver

The receiver is more complicated than the transmitter. This is because of the need to perform symbol timing synchronization and carrier phase synchronization in order to recover from errors between the transmitter and receiver clocks, as mentioned in Section 2.1. The overview of the receiver system is shown in Figure 2.14.

GNU Radio provides a lot of functionality to simplify this system and adjust how both the symbol timing synchronization and carrier phase synchronization are implemented. Figure 2.13 shows the flowgraph used to accomplish everything in Figure 2.14.


UHD: USRP Source
Sync: Unknown PPS
Samp rate (Sps): 250k
Ch0: Center Freq (Hz): 905M
Ch0: AGC: Default
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: RX2

Root Raised Cosine Filter
Decimation: 1
Gain: 1
Sample Rate: 250k
Symbol Rate: 31.25k
Alpha: 350m
Num Taps: 161

Symbol Sync
Timing Error Detector: y[n]y'[n] Maximum Likelyhood
Samples per Symbol: 8
Expected TED Gain: 1
Loop Bandwidth: 45m
Damping Factor: 1.3
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

Carrier Phase Synchronizer
K1: 4.79424m
K2: 5.75309u
mag: 1

QPSK Symbol Decision
symbol magnitude: 1

Fig. 2.13: The first part of the receiver system.

The UHD: USRP Source is nearly identical to the UHD: USRP Sink in terms of parameters. The main difference with this block is it receives rather than transmits. However, the parameters will be the same as described in Section 2.3.1 and should match those of the transmitter, except for the gain value. The RTL-SDR receiver uses its own receiver block known as the Soapy Source, which is very similar to the UHD: USRP Source.

The Root Raised Cosine Filter acts as the matched filter of the system. The parameters of this block (Figure 2.15) should match what was set in the transmitter.

The Symbol Sync block takes care of the symbol timing synchronization, ensuring the symbols are sampled at the correct time. The Symbol Sync block gives a lot of control over the synchronization process (Figure 2.16).

The timing error detector, loop bandwidth, and damping factor are all variable quantities used in tuning the system. This allows the system to get the most accurate symbol samples. Most of the variables were adjustable in between tests, except for the loop bandwidth, which was run-time configurable.

The next block in Figure 2.13 is the Carrier Phase Synchronizer. This is a custom GNU Radio block written to perform the carrier phase synchronization. This block replaces GNU Radio's Costas Loop block in order to better track and correct the phase and frequency offsets caused by clock errors in the devices. This block is controlled by setting $K1$, $K2$

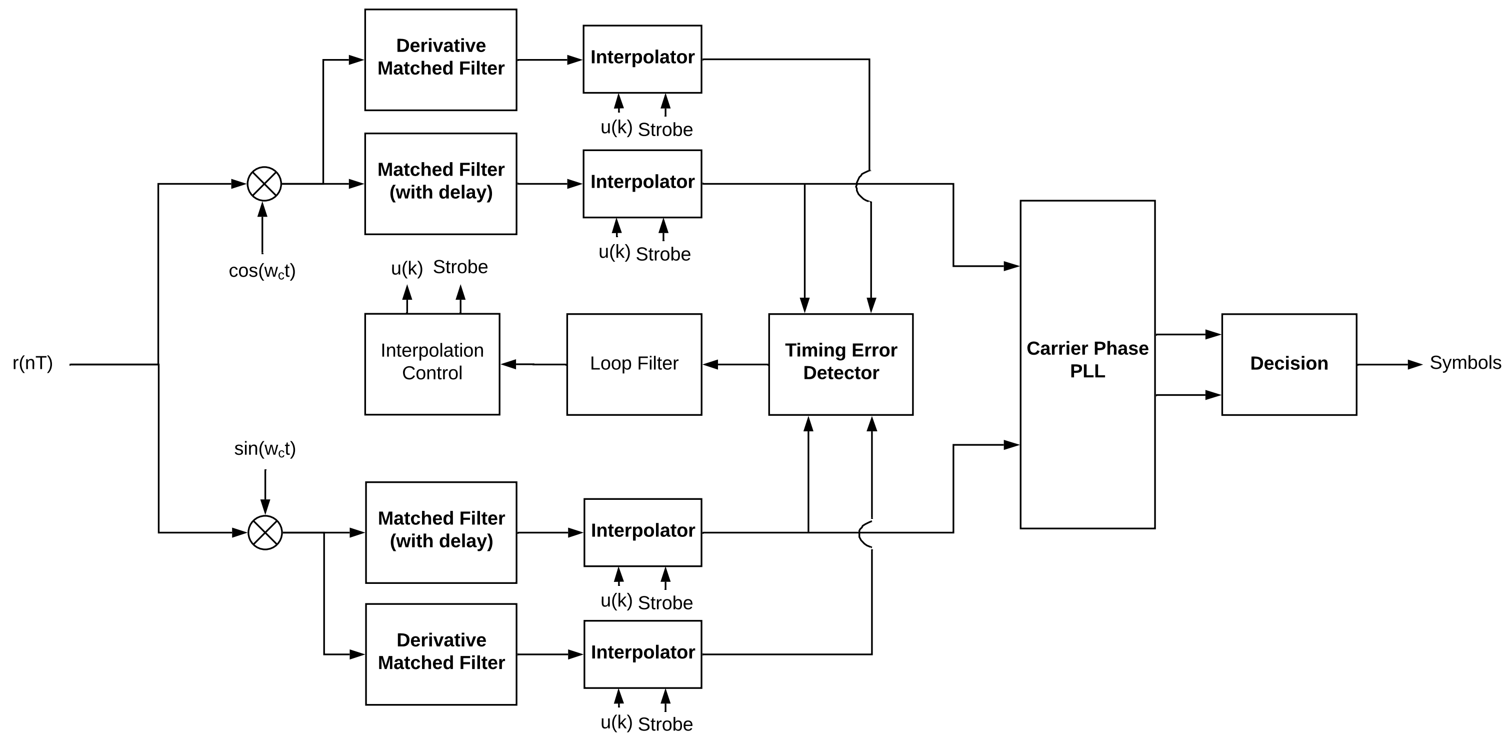
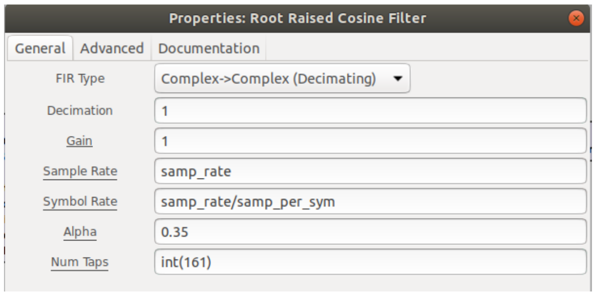


Fig. 2.14: QPSK receiver high level overview.

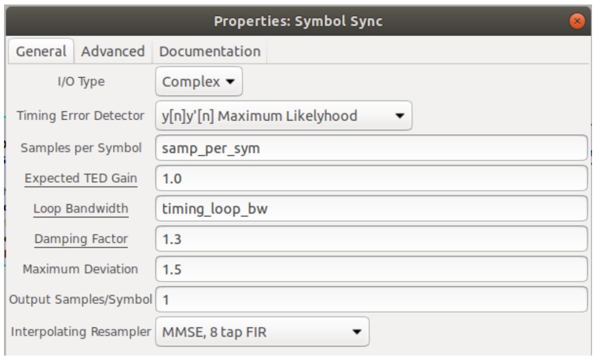


Properties: Root Raised Cosine Filter

General | Advanced | Documentation

FIR Type	Complex->Complex (Decimating)
Decimation	1
Gain	1
Sample Rate	samp_rate
Symbol Rate	samp_rate/samp_per_sym
Alpha	0.35
Num Taps	int(161)

Fig. 2.15: Root Raised Cosine Filter properties.



Properties: Symbol Sync

General | Advanced | Documentation

I/O Type	Complex
Timing Error Detector	y[n]y'[n] Maximum Likelihood
Samples per Symbol	samp_per_sym
Expected TED Gain	1.0
Loop Bandwidth	timing_loop_bw
Damping Factor	1.3
Maximum Deviation	1.5
Output Samples/Symbol	1
Interpolating Resampler	MMSE, 8 tap FIR

Fig. 2.16: Symbol Sync properties.

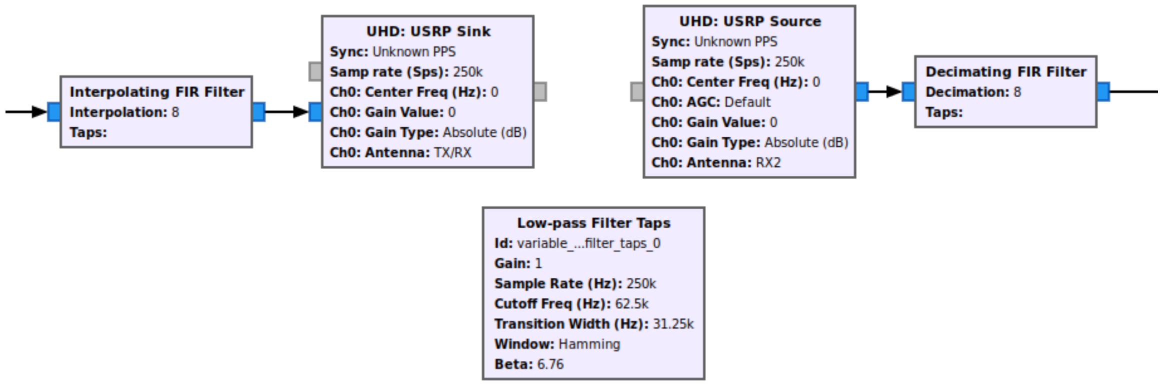
to different values, which controls the loop bandwidth of the loop filter, as calculated in [15, pp. 682]. The magnitude variable should correspond to the magnitude of the QPSK constellation points.

After timing and clock errors are corrected, the symbols are sent into the QPSK Symbol Decision block, where the symbols are estimated to be one of the four constellation symbols in Figure 1.1. After this, the symbols could be used to retrieve the bits transmitted, but this is not necessary for the Doppler extraction.

2.3.3 Hardware Constraints

The previous sections described the general communication system, which will work for some hardware configurations. However, most hardware places a constraint on the sample rate it can communicate at. This constraint comes from the clock rate of the device. Table 2.1 lists a few of the constraints on the sample rate for different devices.

Most of the devices in this table prefer a higher sample rate. It can be hard for the GNU


Interpolating FIR Filter
Interpolation: 8
Taps:

UHD: USRP Sink
Sync: Unknown PPS
Samp rate (Sps): 250k
Ch0: Center Freq (Hz): 0
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: TX/RX

UHD: USRP Source
Sync: Unknown PPS
Samp rate (Sps): 250k
Ch0: Center Freq (Hz): 0
Ch0: AGC: Default
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: RX2

Decimating FIR Filter
Decimation: 8
Taps:

Low-pass Filter Taps
Id: variable_...filter_taps_0
Gain: 1
Sample Rate (Hz): 250k
Cutoff Freq (Hz): 62.5k
Transition Width (Hz): 31.25k
Window: Hamming
Beta: 6.76

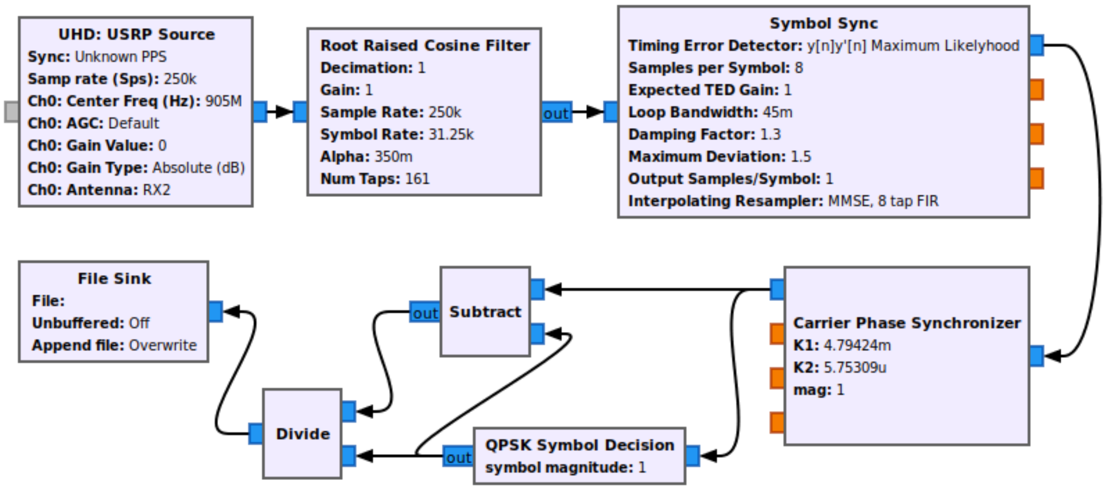
Fig. 2.17: Interpolation and decimation blocks in the transmitter and receiver GNU Radio flowcharts.

Radio receiver system to keep up with this sample rate, depending on the processing power of the computer. Therefore, the GNU Radio communication system can be improved by adding in an Interpolating FIR Filter block before the transmitter block and a Decimating FIR Filter block after the receiver block (Figure 2.17). This allows the sample rate for the hardware to be higher than the sample rate for the GNU Radio transmitter and receiver flowcharts.

2.4 Doppler Extraction

The Doppler frequency extraction was done in the GNU Radio receiver flowchart by adding additional blocks to perform the operations of (2.14) or (2.17), depending on which method was used. Figure 2.18 shows how the Doppler was extracted using the First DEM, while Figure 2.19 shows how the Doppler was extracted using the Second DEM. A File Sink block was also added in order to record the data.

Once GNU Radio extracted the Doppler using either method, the signal was imported into MATLAB and additional signal processing was done to detect and estimate the Doppler frequency path. This processing is the subject of Chapters 3 and 4.


UHD: USRP Source
Sync: Unknown PPS
Samp rate (Sps): 250k
Ch0: Center Freq (Hz): 905M
Ch0: AGC: Default
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: RX2

Root Raised Cosine Filter
Decimation: 1
Gain: 1
Sample Rate: 250k
Symbol Rate: 31.25k
Alpha: 350m
Num Taps: 161

Symbol Sync
Timing Error Detector: y[n]y'[n] Maximum Likelyhood
Samples per Symbol: 8
Expected TED Gain: 1
Loop Bandwidth: 45m
Damping Factor: 1.3
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

File Sink
File:
Unbuffered: Off
Append file: Overwrite

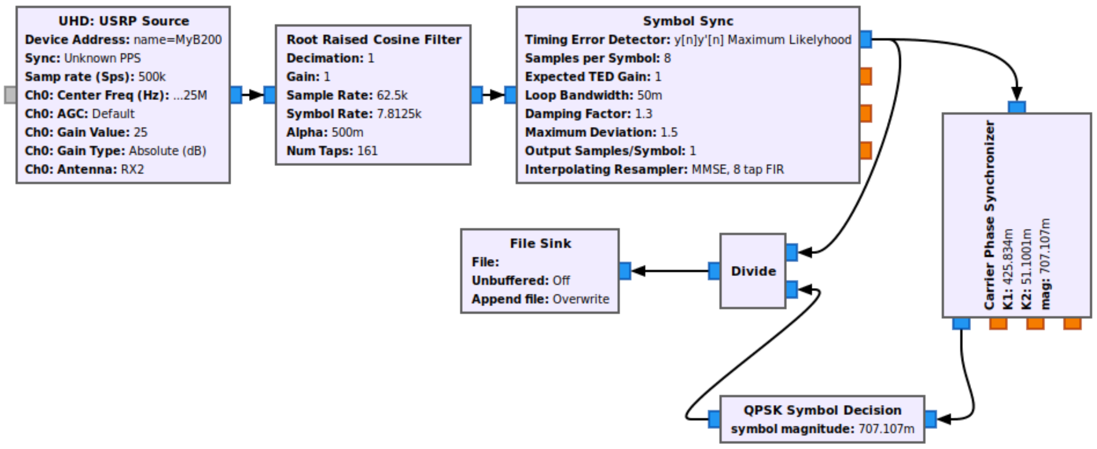
Subtract

Carrier Phase Synchronizer
K1: 4.79424m
K2: 5.75309u
mag: 1

Divide

QPSK Symbol Decision
symbol magnitude: 1

Fig. 2.18: First DEM implemented in GNU Radio.


UHD: USRP Source
Device Address: name=MyB200
Sync: Unknown PPS
Samp rate (Sps): 500k
Ch0: Center Freq (Hz): ...25M
Ch0: AGC: Default
Ch0: Gain Value: 25
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: RX2

Root Raised Cosine Filter
Decimation: 1
Gain: 1
Sample Rate: 62.5k
Symbol Rate: 7.8125k
Alpha: 500m
Num Taps: 161

Symbol Sync
Timing Error Detector: y[n]y'[n] Maximum Likelyhood
Samples per Symbol: 8
Expected TED Gain: 1
Loop Bandwidth: 50m
Damping Factor: 1.3
Maximum Deviation: 1.5
Output Samples/Symbol: 1
Interpolating Resampler: MMSE, 8 tap FIR

Carrier Phase Synchronizer
K1: 425.834m
K2: 51.1001m
mag: 707.107m

File Sink
File:
Unbuffered: Off
Append file: Overwrite

Divide

QPSK Symbol Decision
symbol magnitude: 707.107m

Fig. 2.19: Second DEM implemented in GNU Radio.

CHAPTER 3
SPECTRAL ESTIMATION

The GNU radio communication system is able to remove the symbols from the communication signal using both DEMs of (2.14) and (2.17). Once the symbols are removed, additional signal processing algorithms are used to extract the Doppler signal. This chapter and Chapter 4 explain the signal processing algorithms used to extract the Doppler signal from the output of the GNU radio communication system.

The process of extracting the frequency path consists of transforming the GNU radio output into a spectrogram and then finding the best path through this spectrogram. This chapter discusses methods used to create the spectrogram and improvements made to these methods. These improvements help increase the resolution of the spectrogram and remove the carrier frequency offset and DC components in order to improve the SIR.

The spectrograms discussed in this chapter and the following chapters are plotted in the form of Figure 3.1. The plots are colormap images of magnitude spectra plotted over time. Time progresses downward on the y-axis in the images, and the x-axis shows the frequency bins. These plots are plotted using a dB color scale, with the scale on the right of the image.

Two different methods are employed to create the spectra of the spectrogram: the DFT and the MUSIC algorithm.

3.1 Discrete Fourier Transform

The DFT is the most common way to transform a signal into the frequency domain. It can be implemented using the fast Fourier transform (FFT). The basic equation for the

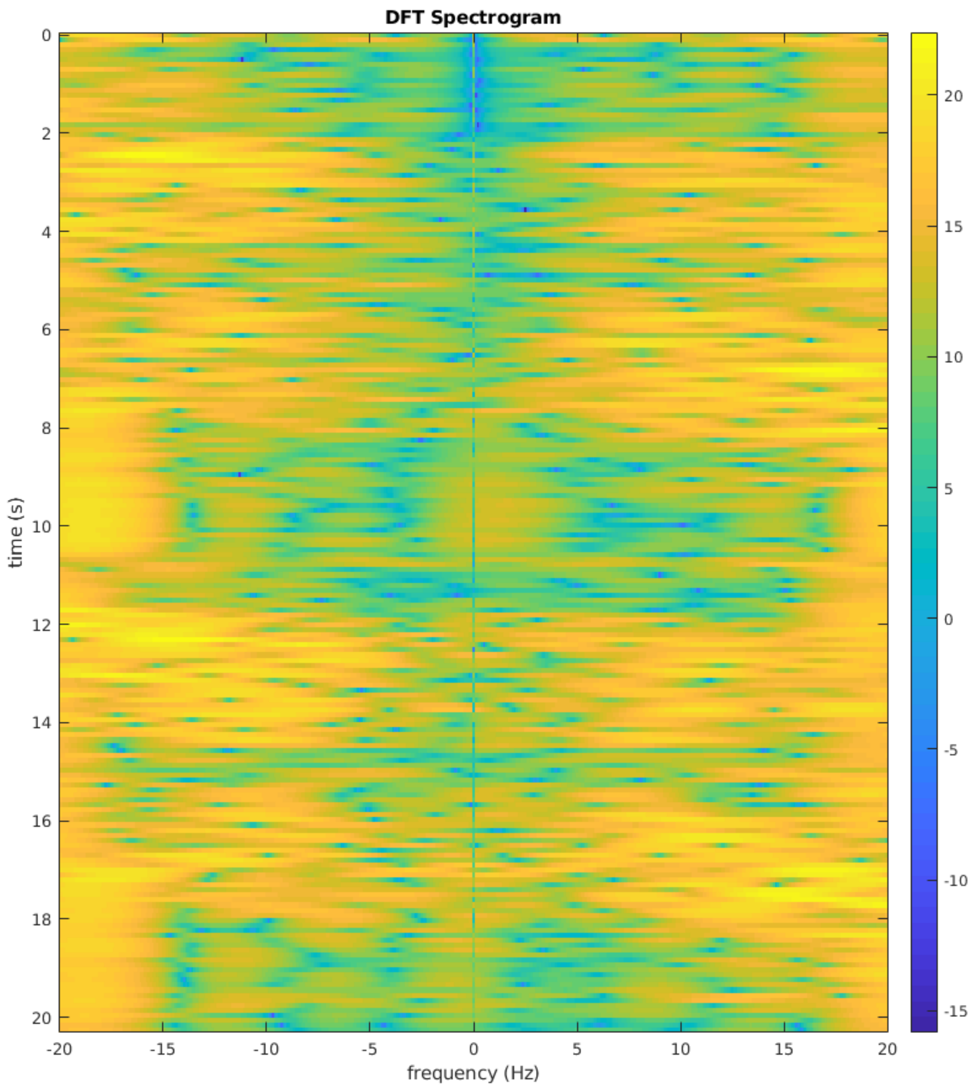


Fig. 3.1: Example of a spectrogram.

DFT is given by [17, Ch. 8]

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] W_N^{kn} \\ W_N &= e^{\frac{-j2\pi}{N}}. \end{aligned} \tag{3.1}$$

This transform takes N samples of data and transforms them into N frequency bins, where frequency bin k represents the actual frequency given by

$$f_k = \begin{cases} \frac{kF_s}{N}, & k \leq N/2 \\ F_s - \frac{kF_s}{N}, & k > N/2. \end{cases} \tag{3.2}$$

F_s is the sample frequency of $x[n]$.

The DFT is simply a sampled version of the DTFT. The DFT samples the DTFT at N points. This means the spectral resolution of the DFT is F_s/N. If N is increased to M by zero-padding the signal to get more bins in the DFT, then the underlying DTFT spectrum will become clearer, as shown in Figure 3.2. However, the underlying spectrum is still the DTFT of the original N signal points, which suffer from windowing effects.

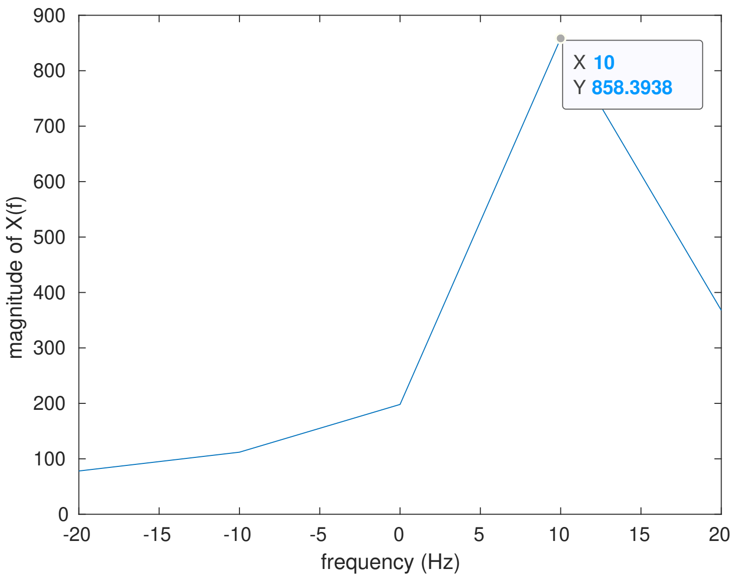
Windowing effects are the results of having limited data. Any signal processed by a computer is considered a windowed signal, as it is limited in length. The basic equation for a windowed signal is

$$x_N[n] = \begin{cases} x[n]w[n], & 0 \leq n < N-1 \\ 0, & \text{otherwise,} \end{cases} \tag{3.3}$$

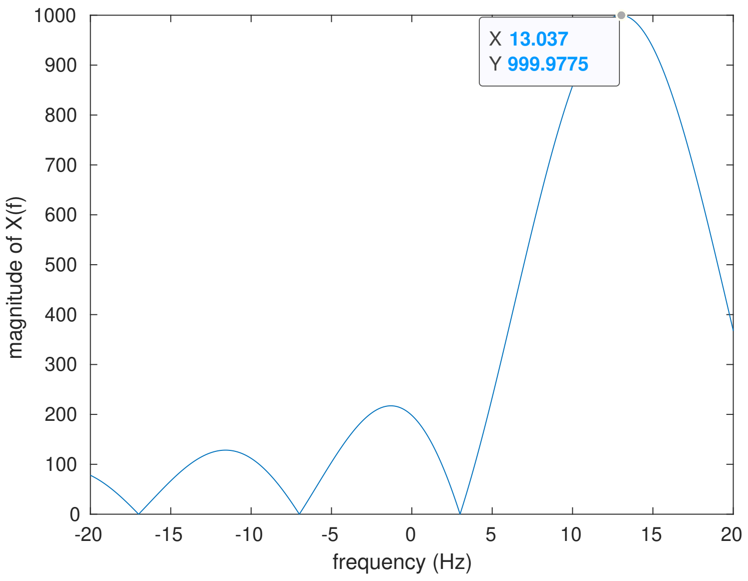
where $x[n]$ is the original signal and $w[n]$ is a windowing function. The simplest windowing function is the rectangular window, $w[n] = 1$. When applying a rectangular window in the frequency domain, this results in convolving the signal with a periodic sinc function in the frequency domain,

$$X_N(f) = X(f) * \frac{\sin(\pi f N)}{\sin(\pi f)} e^{-j\pi f(N-1)}. \tag{3.4}$$

The convolution shifts a copy of the periodic sinc function to each frequency in $X(f)$ and



(a) No zero-padding



(b) Zero padding

Fig. 3.2: Attempts to estimate a signal at 13 Hz with only 10 Hz of resolution.

scales it by the frequency. This means each frequency is represented by a periodic sinc function, as seen in Figure 3.2b.

Each of the periodic sinc functions has a main lobe and side lobes which can interfere with other signals. This interference is known as spectral leakage as each of the frequencies leak into other frequency bins. This is seen in Figure 3.3. In order to limit spectral leakage, a longer window is needed. A longer window decreases the width of the main lobe and decreases the height of the side lobes.

A longer window is a problem for estimating the Doppler frequency. The Doppler frequency changes with time because it varies with the moving object trajectory. The longer the data input into the DFT, the more the Doppler frequency varies over the data resulting in a smearing of frequency content.

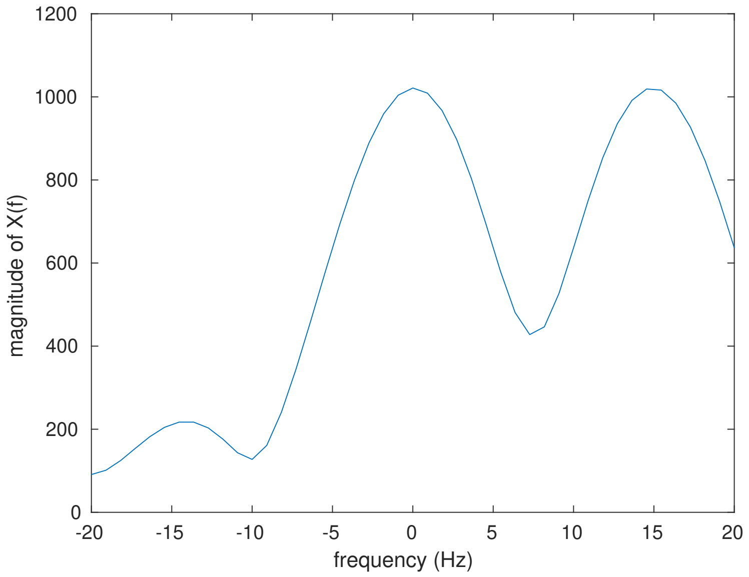
The Doppler frequency for the human tracking application can range from -24 Hz to 24 Hz, as mentioned in Section 1.2.2. It is not unreasonable to assume a human can stop and change direction in about a second, which could cause a lot of smearing if a second of data is input into the DFT.

The trade-off is to find the shortest windowing length possible that keeps the frequency components of interest from being overcome by the spectral leakage of other frequencies in the signal. This can be especially hard in a signal with noise or other large frequency components, like the DC component in the extracted Doppler signal.

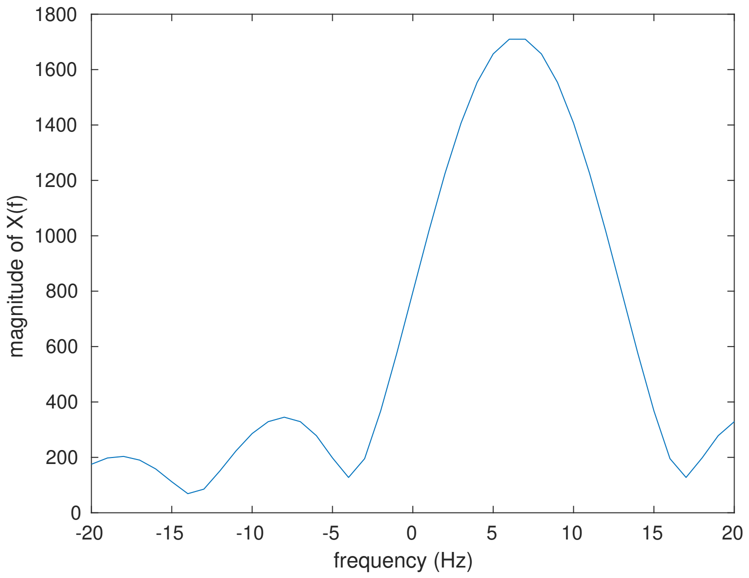
3.2 Multiple Signal Classification

MUSIC is an algorithm capable of improving the spectral resolution of a noisy signal without extending the length of the signal. The MUSIC algorithm establishes a parametric model for the signal as a linear combination of sinusoidal components. By computing the autocorrelation of the data, it is possible to represent the data as a combination of signals in a signal subspace and a noise subspace. This decomposition is valid in situations where the noise is white and has a mean of zero.

The goal of the MUSIC algorithm is to find the noise subspace and then find all of the signals orthogonal to this subspace. The first step to finding the noise subspace is to find



(a) Signals at 0 Hz and 15 Hz



(b) Signals at 5 Hz and 8 Hz

Fig. 3.3: Spectral leakage may result in the loss of the ability to differentiate signals.

the autocorrelation matrix of the data. This matrix is an M by M matrix, with M being an adjustable parameter greater than the number of frequencies in the signal of interest, P. The autocorrelation matrix of the signal $x[n]$ is computed by

$$R_{xx} = \lim_{N\to\infty} \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{x}[n_i]\mathbf{x}[n_i]^*, \tag{3.5}$$

where

$$\mathbf{x}[n_i] = \Big[x[n_i],\ x[n_i+1],\ ...\Big]^T. \tag{3.6}$$

This equation can be approximated with a large N value for finite signals.

The $M - P$ eigenvectors of R_{xx} corresponding to the $M - P$ lowest eigenvalues span the noise space of $x[n]$ [18, pp. 337]. All frequency vectors orthogonal to this noise space are in the signal space. To determine the frequencies, the inner product

$$M(f) = \sum_{k=p+1}^{M} |\mathbf{s}(f)^H \mathbf{u}_k|^2, \tag{3.7}$$

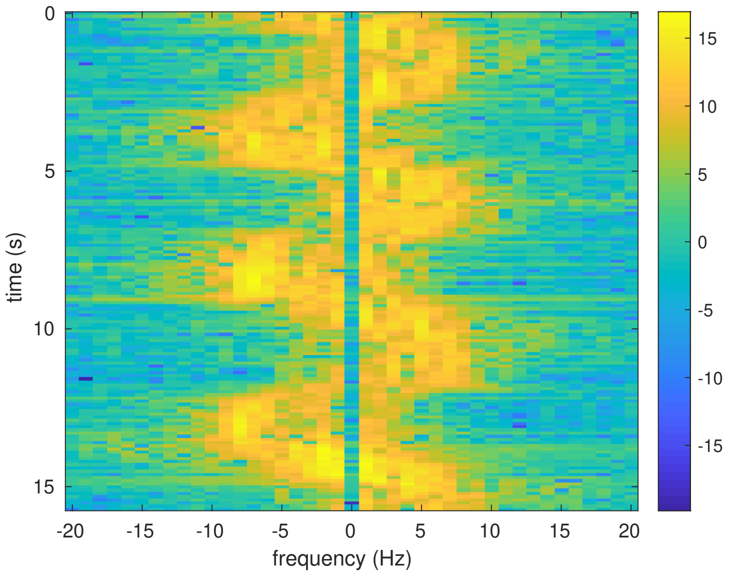
is computed, where

$$\mathbf{s}^T(f) = \Big[1,\ e^{j2\pi f},\ e^{j2\pi 2f},\ ...\ e^{j2\pi(M-1)f}\Big]. \tag{3.8}$$

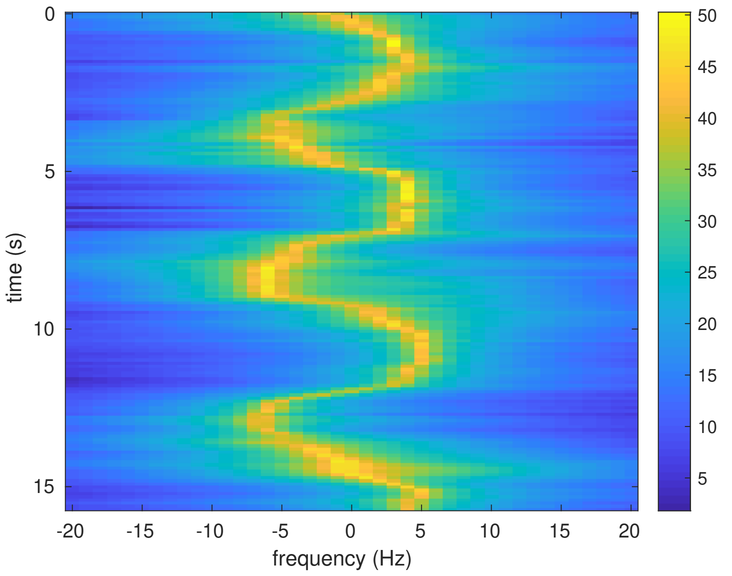
Here, $\mathbf{u}_k$ are the eigenvectors of R_{xx} corresponding to the lowest $M - P$ eigenvalues.

Equation (3.7) computes the inner product between arbitrary frequency vectors and the noise space. The more orthogonal the frequency vector is to the noise space, the closer $M(f)$ is to zero. Plotting the inverse of this function, $P(f) = 1/M(f)$, gives the MUSIC spectrum, which has an appearance similar to the spectrum of the DFT; frequencies in the signal appear as tall slim peaks.

The peaks of MUSIC offer greater spectral resolution without the need for more data. Instead, less data means less accuracy in the computation of the autocorrelation matrix. However, typically MUSIC works better with shorter data lengths than the DFT does. Figure 3.4 shows the difference in the DFT and MUSIC spectrograms when extracting the Doppler frequency path.



(a) DFT spectrogram



(b) MUSIC spectrogram

Fig. 3.4: Comparison of spectrograms generated by the DFT or MUSIC algorithm.

MUSIC works well when the noise is white and zero-mean. When the noise does not fit these assumptions, the MUSIC algorithm gives noisy results. Often, getting a good MUSIC spectrum requires tuning the value of M and sometimes the value of P when it is uncertain how many signals there are.

3.3 Carrier Frequency Offset Removal

This section only applies to the Second DEM of (2.17) as it still has the carrier frequency offset applied to it. As mentioned in Chapter 2, the Second DEM creates a signal with two complex exponential components: one at the carrier frequency offset and one at the carrier frequency offset plus the Doppler shift. Transforming these into the frequency domain would result in two Dirac deltas if both frequencies did not vary with time.

These frequencies do vary with time and, therefore, taking the DFT would require windowing small chunks of the data over a small time step, and doing this over all time steps until a spectrogram is created. Due to limitations with the DFT discussed in Section 3.1 and how small the Doppler shift is, the real DFT plot would contain one large main lobe at the carrier frequency offset, which would cover the Doppler-shifted component (as seen in the spectrogram of Figure 3.5). This means it would be unreasonable to measure both the carrier frequency offset and the Doppler-shifted echo at the same time.

MUSIC would also not be able to separate the two signals because the magnitude of the direct path peak is significantly stronger than the Doppler-shifted peak. The solution is to modulate the signal to baseband and then remove the DC component. Removing the DC is required for both Doppler methods and is discussed in Section 3.4.

Modulating the signal to baseband by removing the carrier frequency offset was initially done in the PLL of the communication system. However, this method resulted in errors because it could not distinguish between the carrier frequency offset and the Doppler frequency.

An alternative method to the PLL is to extract the carrier frequency path from the signal and use this estimated path to demodulate the signal to baseband. This method is not as accurate as using a PLL, but it is able to mostly avoid interference from the Doppler

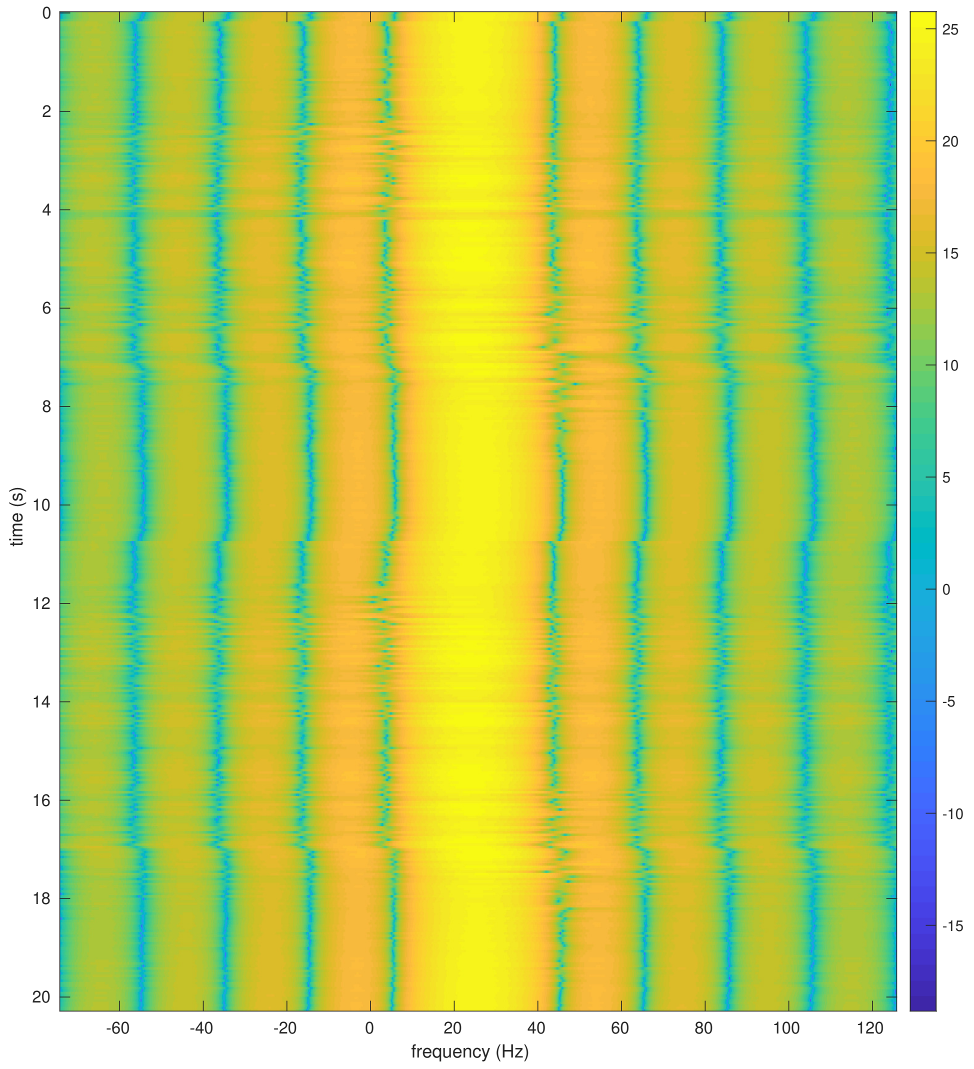


Fig. 3.5: The overwhelming strength of the carrier frequency offset component covers the Doppler frequency.

echo signal. The path extraction techniques mentioned here are discussed more in detail in Chapter 4.

The two main ways to estimate the path of the carrier frequency offset are either to obtain the maximum frequency component at each time step or to use a Viterbi algorithm to find the most likely path.

Obtaining the maximum frequency component at each time step is computed by finding the maximum value of the magnitude spectrum. The frequency of this maximum value is the carrier frequency offset over this window of data. With this frequency known, the window of data used for generating the spectrum can be modulated to baseband. Then it can be combined with the previously modulated data windows.

Finding the maximum works when the Doppler frequency component is too weak to interfere with the carrier frequency offset. However, this is not true for the entirety of most signals. The amplitude of the Doppler echo changes over the course of the signal, and there are instances when the Doppler echo will influence this maximum value. Figure 3.6 shows the Doppler frequency causing small changes in the main lobe of the carrier frequency offset.

These effects can pull the carrier frequency offset estimate from the true value, which would lead to errors in modulating the signal to baseband. The Viterbi algorithm is an alternative method capable of ignoring these slight perturbations. This algorithm is discussed in Section 4.1.

The Viterbi algorithm is more complex but more accurate in estimating the carrier frequency offset. However, once the signal is at baseband, there is still a DC component to remove.

3.4 DC Component Removal

Estimating the Doppler signal with limited data is difficult when there is a large DC component in the signal. Using either DEM in (2.14) or (2.17) (once the carrier frequency offset is removed) will result in an output having a large DC component, usually much larger than the Doppler frequency component.

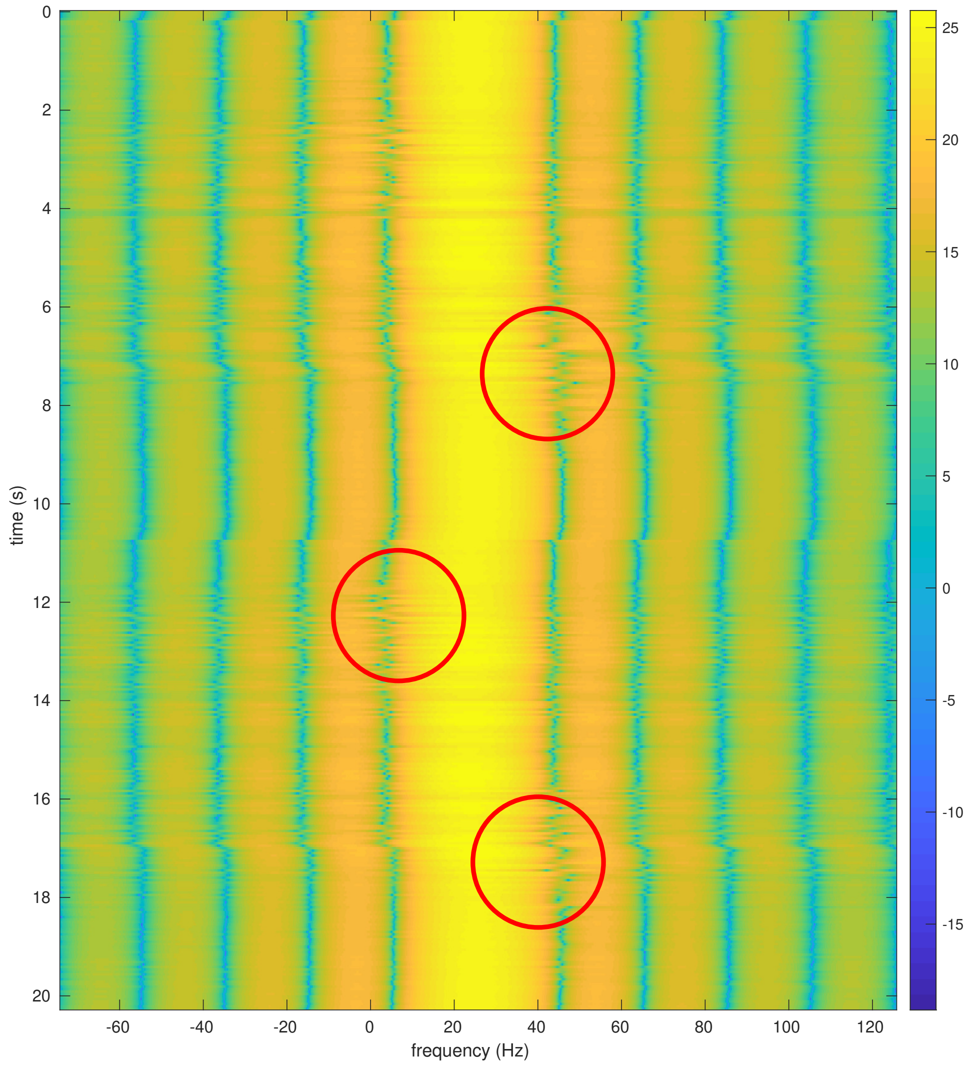


Fig. 3.6: The Doppler frequency causing changes to the main lobe of the carrier frequency offset path.

This component is a result of interference from the direct path signal. This interference, as mentioned in Section 1.1, is a limiting factor to the system. This component causes the Doppler frequency to be overpowered by spectral leakage when the DFT is calculated and causes problems in calculating the MUSIC algorithm, as the MUSIC algorithm expects the noise to have a mean of zero. Therefore, it is necessary to remove the DC component.

Removing the DC component turns out to be trivial as it can be accomplished by subtracting the mean of the signal. It is important to subtract this mean at the correct place, which differs for both DEMs. For the First DEM of (2.14), the mean can be removed either over the whole signal of length D,

$$\mathrm{DEM}_{1,\text{ zero-mean}}(k) = \mathrm{DEM}_1(k) - \frac{1}{D}\sum_{i=0}^{D-1} \mathrm{DEM}_1(i), \tag{3.9}$$

or separately in each window of data used to generate the DFT or MUSIC algorithm,

$$\mathrm{DEM}_{1,\text{ zero-mean}}(k) = \mathrm{DEM}_1(k) - \frac{1}{N}\sum_{i=n}^{N-1} \mathrm{DEM}_1(i), \qquad \text{for } n \le k < N. \tag{3.10}$$

Typically, subtracting the DC in each window of data works better. For the Second DEM of (2.17), the best place to subtract the mean is right after modulating a piece of the signal to baseband but before combining the signals together,

$$\mathrm{DEM}_{2,\text{ zero-mean}}(k) = \mathrm{DEM}_2(k)e^{j2\pi \hat{f}_c k} - \frac{1}{N}\sum_{i=n}^{N-1} \mathrm{DEM}_2(i)e^{j2\pi \hat{f}_c i}, \quad \text{for } n \le k < N. \tag{3.11}$$

CHAPTER 4
PATH EXTRACTION

The previous chapter dealt with generating a spectrogram from the extracted Doppler signal. This chapter discusses methods of extracting the best Doppler frequency path from the spectrogram. This chapter will represent the optimal path extracted by the path-finding algorithms as red asterisks on top of the spectrogram.

The simplest extraction method is the maximum value extraction. This method simply finds the maximum value of the spectrum at each time step and records the frequency at the maximum value. This list of recorded frequencies is the Doppler frequency path. This method is the same as the carrier frequency maximum extraction method described in Section 3.3.

The maximum value extraction method works in the ideal case where there is no noise or direct path interference on the signal or where both the SNR and SIR are large. However, the Doppler echo is a weak signal, which the noise or direct path interference can easily bury. This results in the estimated Doppler frequency path becoming noisy and unreliable. Figure 4.1 shows the maximum value extraction applied to a MUSIC spectrogram.

The maximum value extraction is not the best way to extract the Doppler path due to these limitations. The rest of this chapter is dedicated to exploring more reliable ways of extracting the Doppler path.

4.1 Viterbi Algorithm

The Viterbi algorithm is typically used in error correction coding to decode convolutional codes [19, pp. 471]. The basic premise of the algorithm is to find the best path through a system with a discrete number of states. Typically, a trellis diagram, like the one in Figure 4.2, is used to depict the operation of the algorithm.

This algorithm is able to find a connected path through the states of the trellis by

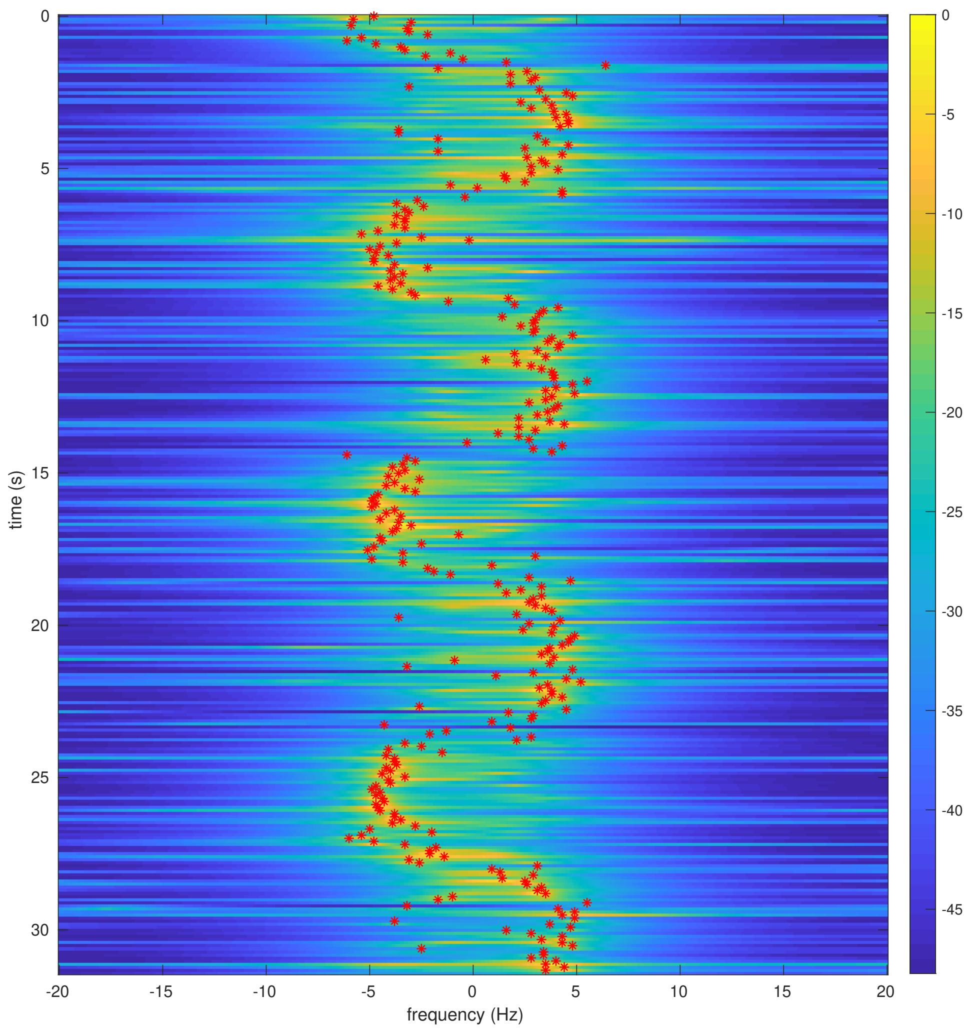


Fig. 4.1: Maximum value path extracted from a MUSIC-generated spectrogram.

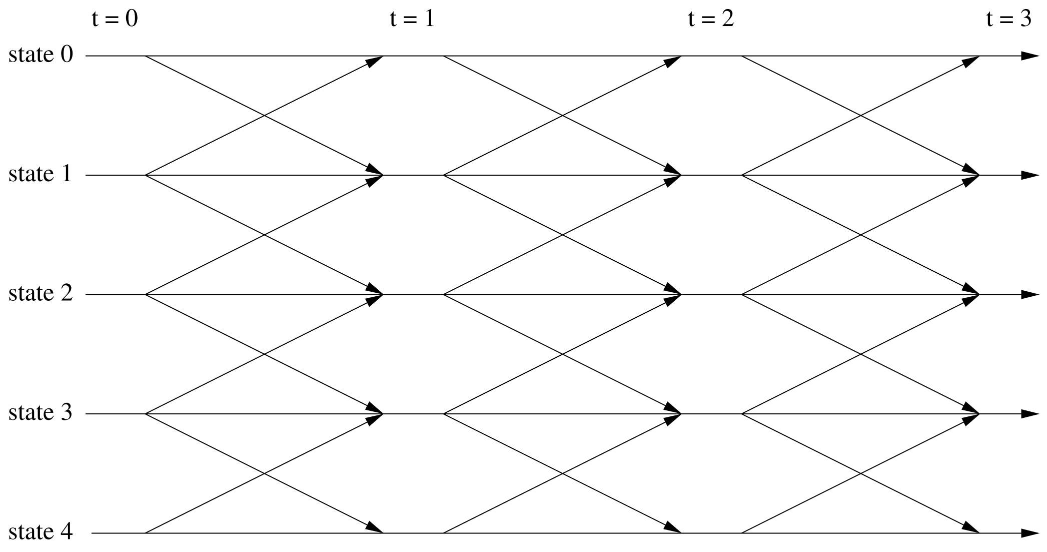


Fig. 4.2: Trellis diagram of a Viterbi algorithm.

putting limits on the state transitions. This makes it easier to ignore sudden jumps to far away states, which makes the system less noise prone. This algorithm follows the best path to each final state and then proceeds to select the best state overall as the most likely path through the trellis.

The Viterbi algorithm finds the best path through the trellis by finding the path with either the lowest path cost or the highest path value. The application determines whether the algorithm uses the lowest path cost or highest path value. For finding the best Doppler frequency path, the highest path value is used for optimizing the path.

To find the best path through the spectrogram, the algorithm uses the frequency bins of the spectrogram as the different states of the trellis. Each state is then limited in the number of states it can transition to. This is shown in the trellis of Figure 4.2 as a limited number of paths from one state to another state. In this case, each state in this trellis can only transition to either itself or one of the neighboring states.

For the frequency bins, this limit is based on how far the Doppler frequency might change from time step to time step. For example, in the human tracking application observations show that Doppler frequency does not change by more than 3 Hz over a 0.1 second period. Therefore, the trellis should not allow any of the states to transition to states more than 3 Hz away. This limit in frequency shift defines the trellis structure. The limit of how far away in Hz a state may transition to is denoted by δ.

Once the structure is set, the algorithm then iteratively runs through the states of the trellis. The algorithm begins at $t = 0$ by initializing each state's path value. This initialized value is the magnitude of the spectrum at $t = 0$ for each frequency bin. Then the algorithm moves to $t = 1$.

At this next time step, a new path value is generated for each transition to another state. At each new state, the transition with the highest path value to the new state is kept while all other transitions are ignored. This new value becomes the updated score for this state, and the transition is recorded in a transition vector.

This method is then repeated for each additional time step after $t = 1$. At $t = n$ the path value is generated for each transition to state k and then the best value is kept and the corresponding transition is recorded. This means at any time there is a score for each state and a transition vector to each state describing the most optimal path to the state.

The path value for state k at time $t = n$, $M_n(k)$, is calculated using

$$M_n(k) = \max_j [M_{n,n-1}(k, j)], \tag{4.1}$$

where

$$M_{n,n-1}(k, j) = M_{n-1}(j) + v(k) - \mu(k, j) \tag{4.2}$$

refers to the path value of state j of the previous time step that is transitioning to state k of the current time step. Only the best of these transitions are kept.

The path value $M_{n,n-1}(k, j)$ depends on the previous path value of state j, $M_{n-1}(j)$, and the branch metric. The branch metric is composed of the magnitude of the frequency bin of state k,

$$v(k) = abs(X[k = f/N]), \tag{4.3}$$

and a transition penalty,

$$\mu(k, j) = K|k - j|. \tag{4.4}$$

The transition penalty is an important parameter for limiting transitions to other frequencies. The further away the frequency the more this parameter lowers the path value. This parameter is controlled by a variable K.

The K value is a weight useful for adjusting the algorithm. The higher K is, the more expensive it is to change frequencies in the algorithm. This discourages the Viterbi algorithm from tracking large shifts in frequency due to a high spike of noise, but it will still track a change in Doppler.

In addition to K, δ is the other variable for controlling the maximum allowed frequency transition. This value controls how quickly the path is able to change, as mentioned previously. Together, these variables ensure the Viterbi algorithm is properly tuned to avoid noise. Figure 4.3 shows the Viterbi algorithm applied to the same spectrogram as in Figure 4.1. The path in Figure 4.3 is less noisy than the path in the maximum value case. This is due to the smoothing ability of the Viterbi algorithm. However, the Viterbi algorithm can smooth the data too much if incorrectly adjusted.

4.2 BCJR Algorithm

The Viterbi algorithm improves the Doppler path by using previous data to better predict more recent data. However, it only uses past and present data at each time step to determine the most likely path. Using future data of the state of the system may improve the results. BCJR is an algorithm capable of determining the best path using past, present, and future data.

BCJR generates a matrix of probabilities by assigning a probability to each state at every time step based on past, present, and future information. This limits the real-time application of this algorithm as it requires future data. However, this algorithm can be implemented in real-time if the application allows some latency in extracting the Doppler path. BCJR comes from the field of error correction coding where it is used in decoding turbo codes [19, pp. 588].

The equations for BCJR are listed as (4.5) – (4.7) [19, pp. 596]. In these equations, p refers to the state at time t, while q refers to the state at time $t+1$.

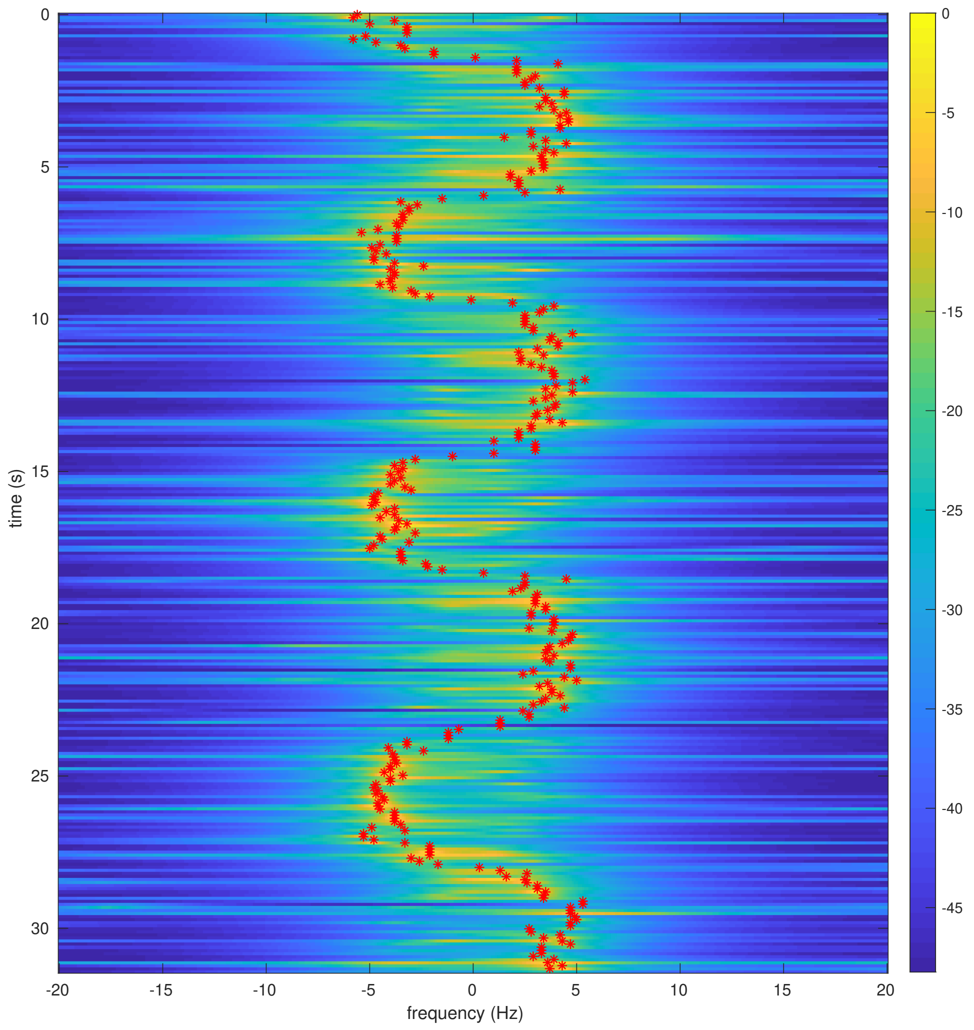


Fig. 4.3: Path extraction using the Viterbi algorithm through a MUSIC generated spectrogram.

$$\alpha_{t+1}(q) = N_q \sum_{p=0}^{Q-1} \alpha_t(p)\gamma_t(p,q) \tag{4.5}$$

$$\beta_t(p) = N_p \sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_{t+1}(q) \tag{4.6}$$

$$P(x|\boldsymbol{r}) = \alpha_t(p) \sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_{t+1}(q) \tag{4.7}$$

Equation (4.5) is referred to as the forward pass equation. This recursive equation analyzes how the previous states and current state affect future states. The $\alpha_0(p)$ values are initialized to $1/Q$ where Q is the number of states (it is equally likely to start in any of the states). Equation (4.6) is referred to as the backward pass equation. This equation analyzes how present and future states affect the past states. The $\beta_{N-1}(q)$ values, with N being the total number of time steps, are initialized to $1/Q$ (it is equally likely to end in any state). N_q and N_p are normalization constants and are defined as

$$N_q = \frac{1}{\sum_{q=0}^{Q-1} \sum_{p=0}^{Q-1} \alpha_t(p)\gamma_t(p,q)} \tag{4.8}$$

$$N_p = \frac{1}{\sum_{p=0}^{Q-1} \sum_{q=0}^{Q-1} \gamma_t(p,q)\beta_t + 1(q)}. \tag{4.9}$$

Equation (4.7) takes the results from the forward and backward passes to calculate the probability of the state x given all of the observations $\boldsymbol{r}$.

$\gamma_t(p,q)$ describes the transition between states. This value takes into account the magnitude of the frequency spectrum, $v(f)$, and the shift in frequency from state p to q. For the purpose of extracting the Doppler signal, $\gamma_t(p,q)$ is

$$\gamma_t(p,q) = \begin{cases} e^{K_2 v(p)} e^{-K_1|p-q|}, & \text{if } |p-q| < \delta \\ 0, & \text{otherwise.} \end{cases} \tag{4.10}$$

This equation has three parameters for adjusting the algorithms, K_1, K_2, and δ. K_1 is the weight given to the branch metric, which is the same as the branch metric in the Viterbi algorithm. K_2 is the weight given to the magnitude of the frequency spectrum for the current state. δ limits how many states a single state can transition to, which is the same as δ in the Viterbi algorithm.

As mentioned, the BCJR algorithm outputs a matrix of probabilities the same size as the input spectrogram matrix. The maximum value path extraction method can extract the best frequency path, but it is typically better to use the Viterbi algorithm.

The Viterbi algorithm is preferred for finding a path in the BCJR probability matrix for the same reasons it is preferred to find a path in the spectrograms: it finds a more continuous path. The BCJR algorithm is not able to smooth the path as much as the Viterbi algorithm does, as it does not look for a path. However, the BCJR algorithm does do a good job improving the values for the Viterbi algorithm to work on.

Figure 4.4 shows the results of using the BCJR algorithm with the Viterbi algorithm. This figure displays the path on top of a color map of the probability matrix generated by BCJR from the MUSIC spectrum of Figure 4.1.

4.3 Balance Factor

The Viterbi and BCJR algorithms are able to improve the path extraction in most cases. However, they fail when the Doppler frequency, $f_d(t)$, is at 0 Hz. When the Doppler frequency is at this DC term, the path extraction is unable to find it.

This occurrence is not uncommon, as it occurs anytime there is no movement or when a moving target moves directly between the transmitter and receiver, as indicated by the bistatic radar equation of (1.10). The derivative terms in (1.10) cancel out when they are equal in magnitude but opposite in sign. This occurs anytime the target is moving directly towards the receiver and directly away from the transmitter or directly away from the transmitter and directly towards the receiver. At this moment, $f_d(t) = 0$, the same as the direct path signal. The processing required to remove the direct path signal (DC removal of Section 3.4) also removes the Doppler frequency in this case.

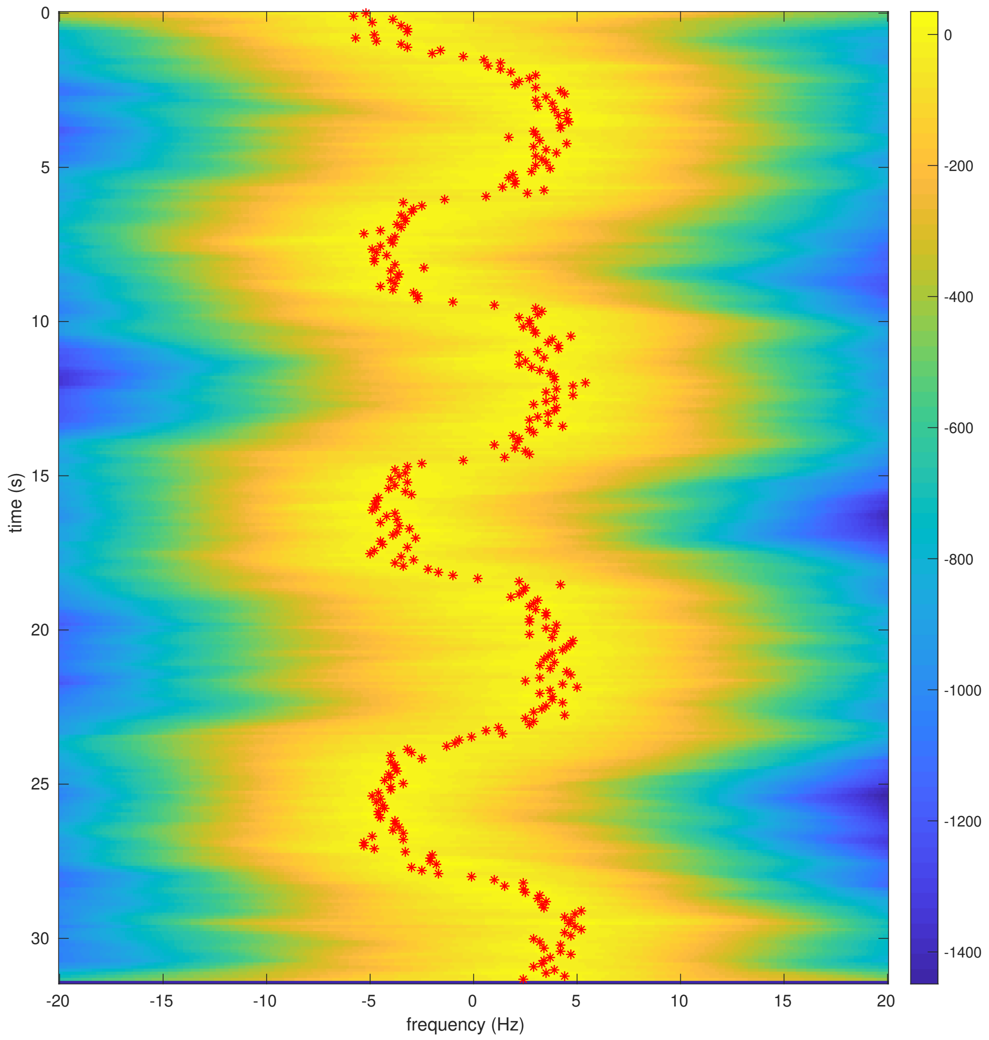


Fig. 4.4: Path extraction using a Viterbi algorithm on a BCJR probability matrix formed using a MUSIC generated spectrogram.

If $f_d(t)$ merely passes through 0 Hz, then the path extraction algorithms are able to easily correct for the lost data. However, when $f_d(t)$ stays at 0 Hz for an extended period of time, the algorithms will try to find a path in the noise. This leads to erroneous Doppler frequency data.

A solution to this is to introduce the idea of a balance factor. A balance factor is a ratio describing how the energy in a spectrum is balanced between the positive and negative frequency bins. It is calculated simply by computing the sum of the magnitudes of all the positive frequencies of the spectrum and then subtracting the sum of the magnitudes of all the negative frequencies. Then the result is normalized by the sum of the magnitude of all frequencies.

$$BF(t) = \frac{\sum_{0}^{\max f} |v_t(f)| - \sum_{\min f}^{0} |v_t(f)|}{\sum_{\min f}^{\max f} |v_t(f)|} \tag{4.11}$$

The resulting balance factor is between -1 and 1. This value shows how balanced the spectrum is. When calculated for each spectrum in the spectrogram, the result is a vector of these balance factors. Plotting these factors and scaling them shows how they follow the Doppler frequency to either side and hover around 0 when the Doppler frequency is around 0 Hz, as shown in Figure 4.5.

The balance factor contains the information needed to make sure the path extraction knows when the Doppler is at or near 0 Hz. The Viterbi and BCJR algorithms can use this factor to influence their decisions. The easiest way to implement this factor in these algorithms is to force the path to 0 Hz anytime $|BF(t)| < \epsilon$, where ϵ is a threshold parameter. This approach is error prone, as noise heavily influences the balance factor. Therefore, if the noise pushes the balance factor below ϵ while the Doppler is actually at 10 Hz, the results are a sudden jump to 0 Hz and then back to 10 Hz.

A better approach is to pull the spectrum to 0 Hz when $|BF(t)| < \epsilon$. This was implemented in the BCJR algorithm by adding an additional term to $\gamma(t)$ in (4.10), although a similar adjustment could also have been made to the Viterbi algorithm in (4.2). This

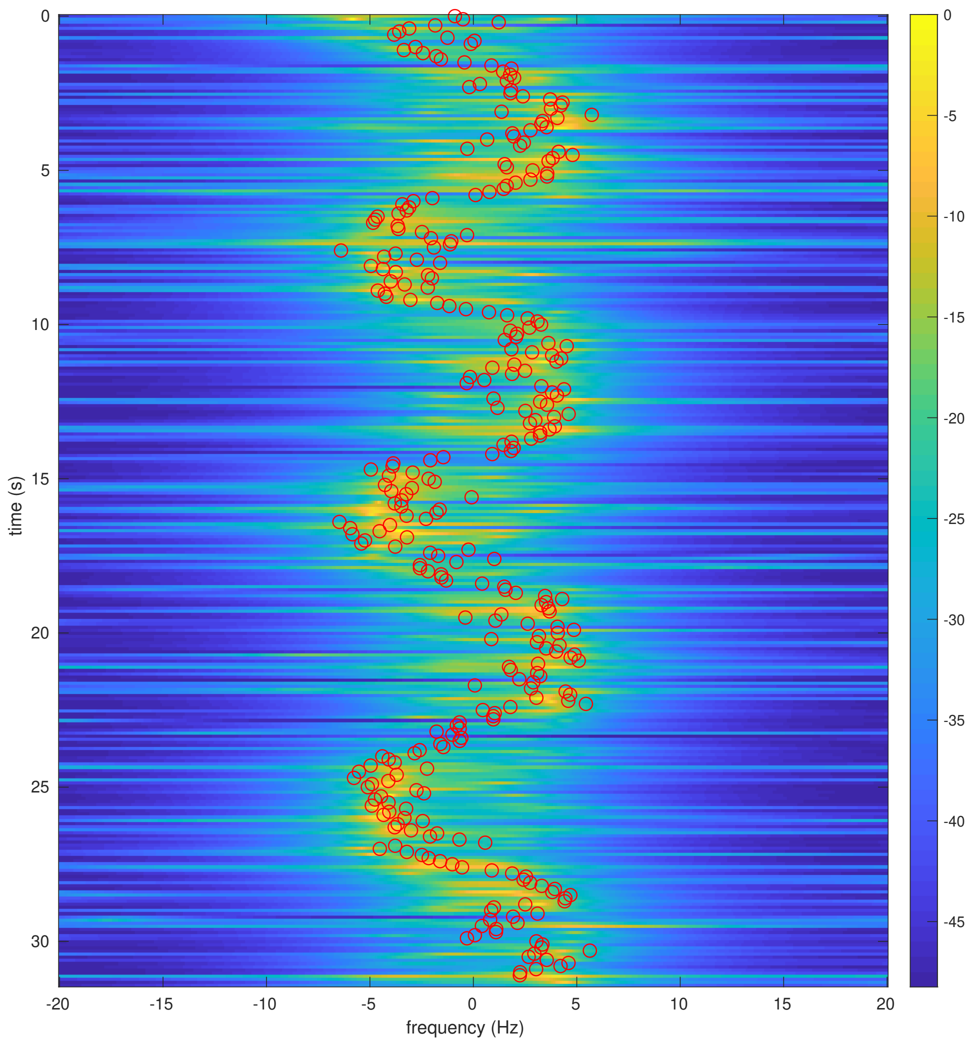


Fig. 4.5: A scaled balance factor vector plotted on top of the MUSIC spectrum the balance factor represents.

adjustment results in

$$\gamma_t(p,q) = \begin{cases} e^{K_2 v(p)} e^{-K_1|p-q|}, & \text{if } |p-q| < \delta \text{ and } |BF(t)| > \epsilon \\ e^{K_2(\tilde{B}F(t)v(p)+(1-\tilde{B}F(t))v_0(p))} e^{-K_1|p-q|}, & \text{if } |p-q| < \delta \text{ and } |BF(t)| \le \epsilon \\ 0, & \text{otherwise,} \end{cases} \tag{4.12}$$

where

$$\tilde{B}F(t) = K_0|BF(t)|$$

and

$$v_0(p) = \begin{cases} \Lambda - |K_3 p|, & \text{if } |p| < \sigma \\ 0, & \text{otherwise.} \end{cases}$$

This adjustment uses the $BF(t)$ term as a weight to adjust the shape of the magnitude spectrum when $BF(t)$ is below the threshold value ϵ. This adjustment is a balance between the magnitude spectrum and an adjustment spectrum, such as the one depicted in Figure 4.6. This adjustment spectrum gives more weight to the lower frequencies when the balance factor is closer to 0.

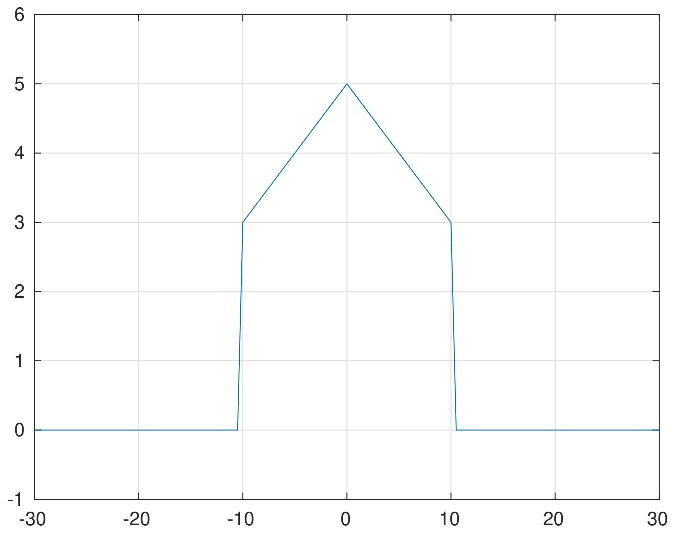


Fig. 4.6: Adjustment frequency spectrum used to pull the Doppler frequency closer to 0 Hz.

This new equation adds several new parameters. K_0 is a parameter used to control the weight given to $BF(t)$. This weight is necessary, as, in practice, $BF(t)$ is generally small and can give too much weight to the adjustment spectrum. The parameters Λ, ϵ, and K_3 all control the shape of the adjustment spectrum, $v_0(p)$: Λ controls the height of the spectrum, ϵ controls the width of the spectrum, and K_3 controls the rate at which the magnitude decreases linearly away from the maximum at 0 Hz. This allows the spectrum to either form a triangle or an irregular pentagon.

Applying this adjustment allows the spectrum to account for Doppler at 0 Hz. Figure 4.7 shows the BCJR probability plot and optimal path when trying to identify the optimal path from a spectrogram with portions of the signal at 0 Hz. This figure demonstrates that the optimal path is not able to sufficiently transition from side to side with the Doppler frequency. Figure 4.8 shows the results after applying the balance factor adjustments to the algorithm. The adjustments allow the BCJR to give higher probabilities to frequencies closer to 0 Hz, as seen around the 10-second mark in Figure 4.8. This also makes it less likely to be on the wrong side as the balance factor makes transitions from side to side more probable.

4.4 Correcting Symmetry

This section applies only to the First DEM of (2.14). Section 2.1.1 discussed how this method breaks down in the carrier phase PLL, as the PLL tries to track the Doppler frequency and remove it. This results in the generation of a new frequency term at the negative Doppler frequency (shown in (2.15)). This symmetric frequency problem renders the First DEM useless when the carrier phase PLL is used unless additional processing is done to recover the sign of the Doppler frequency. Therefore, a method to extract the sign of the Doppler frequency was devised. This method takes advantage of the rotations of the symbols around the constellation points to determine the direction of rotation. Figure 4.9 shows a diagram of how the Doppler frequency modifies the signal constellation by rotating the symbols.

There are two important parameters to keep track of in relation to these rotations: the

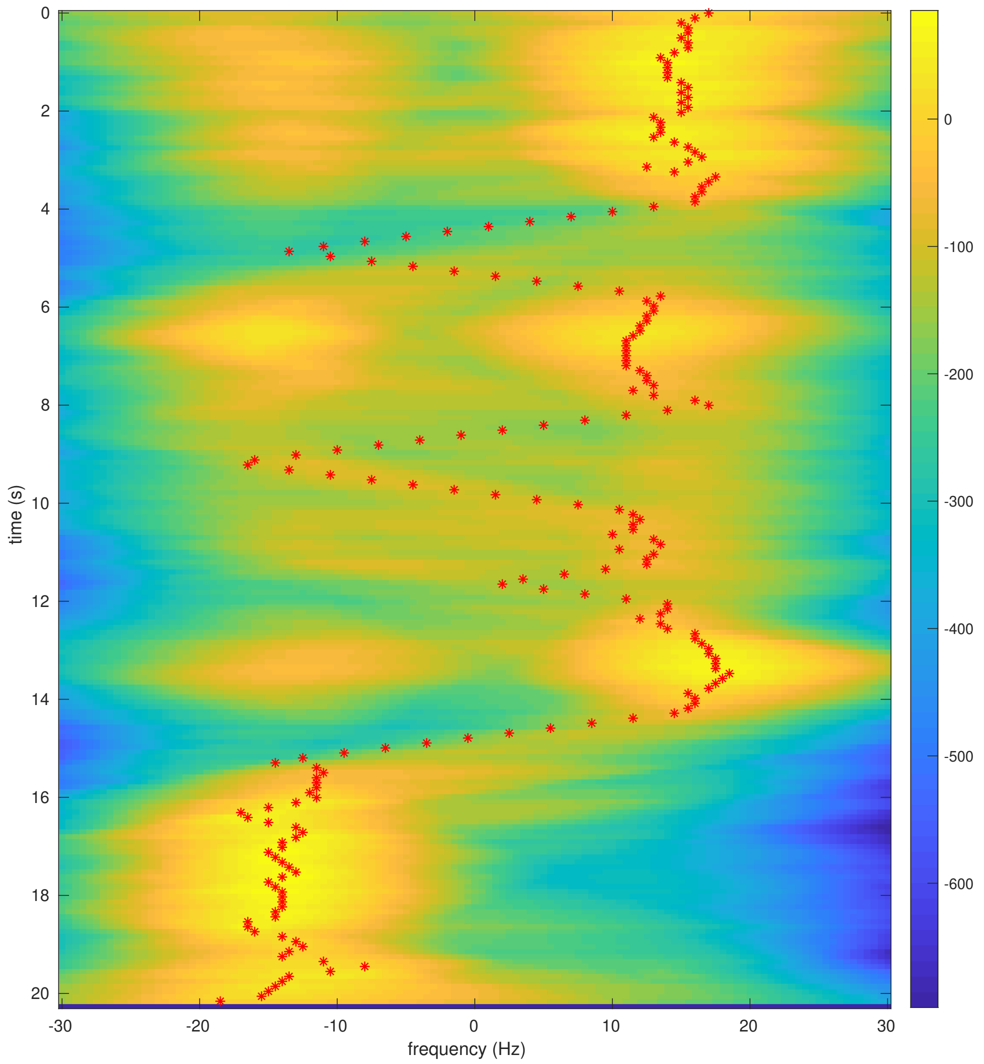


Fig. 4.7: Trying to apply the BCJR algorithm to a spectrogram without the balance factor adjustment when there are portions of the signal at 0 Hz.

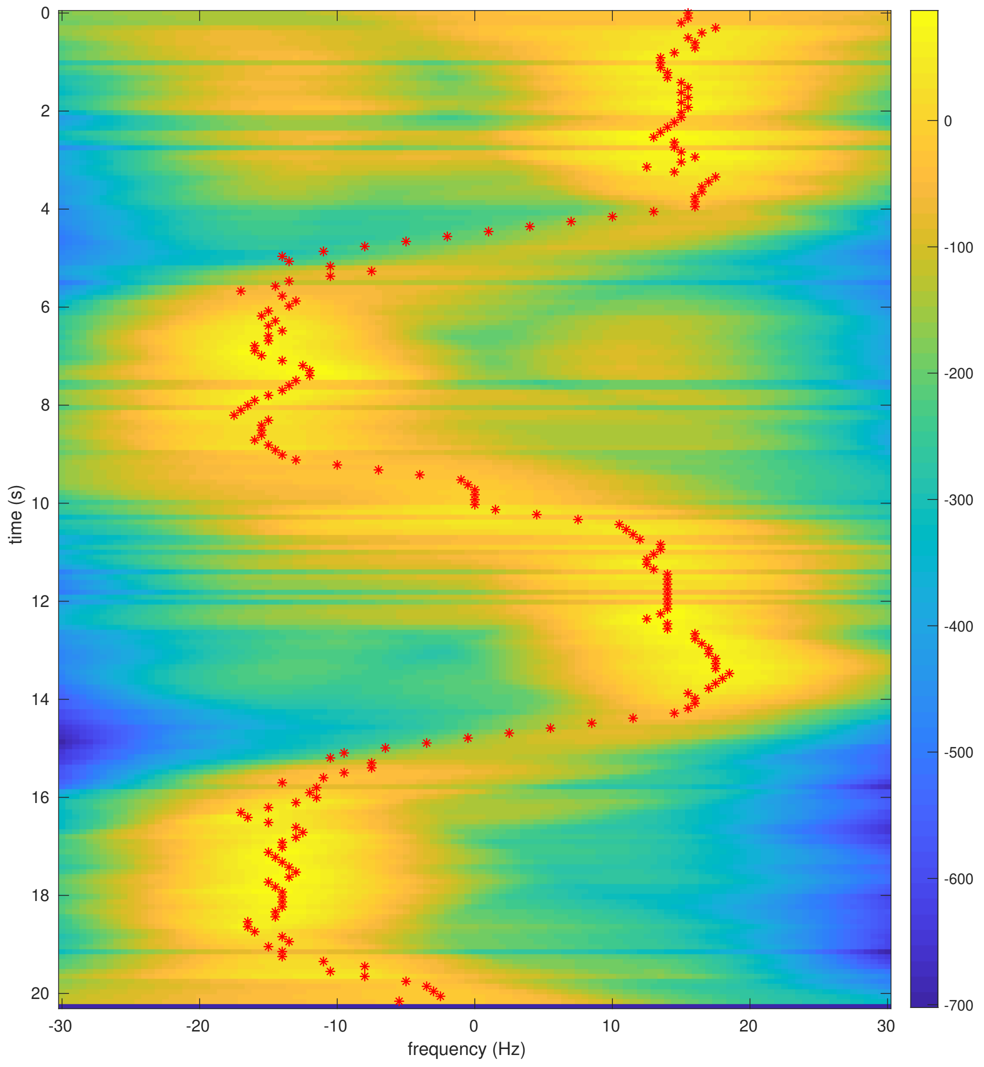


Fig. 4.8: Trying to apply the BCJR algorithm to a spectrogram with the balance factor adjustment when there are portions of the signal at 0 Hz.

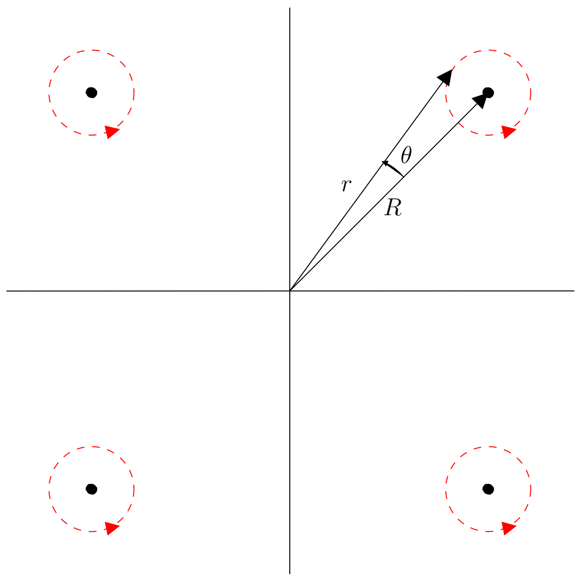


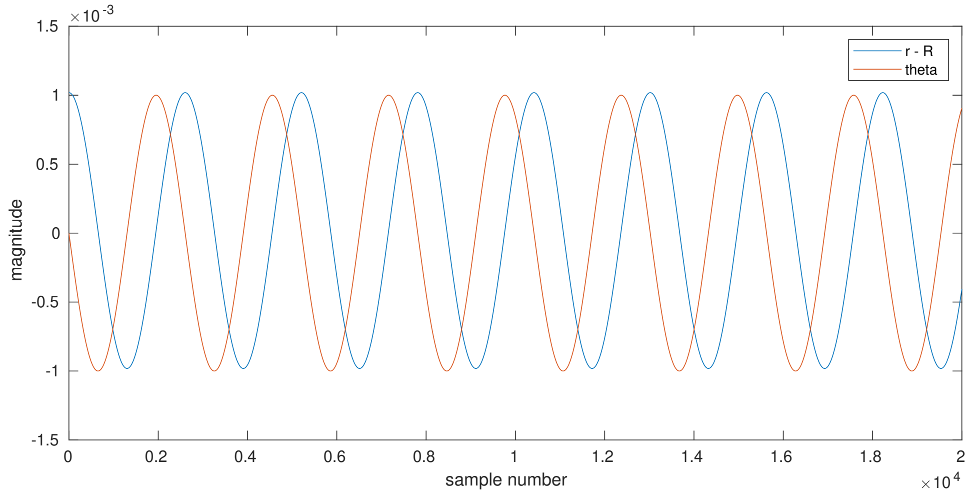
Fig. 4.9: Diagram of how Doppler echo symbols rotate on a non-rotating signal constellation.

angle of error, θ, and the radius of error, r. These two quantities define how the Doppler frequency modifies the signal point. If the symbols are all a single constellation point, a plot of θ compared to the value of $r - R$, with R being the average radius, would result in the plots of Figure 4.10. If the symbols are from multiple constellation points the plots would be the same as long as θ is measured in comparison to the corresponding constellation point, which would be obtained from the estimated symbol data.

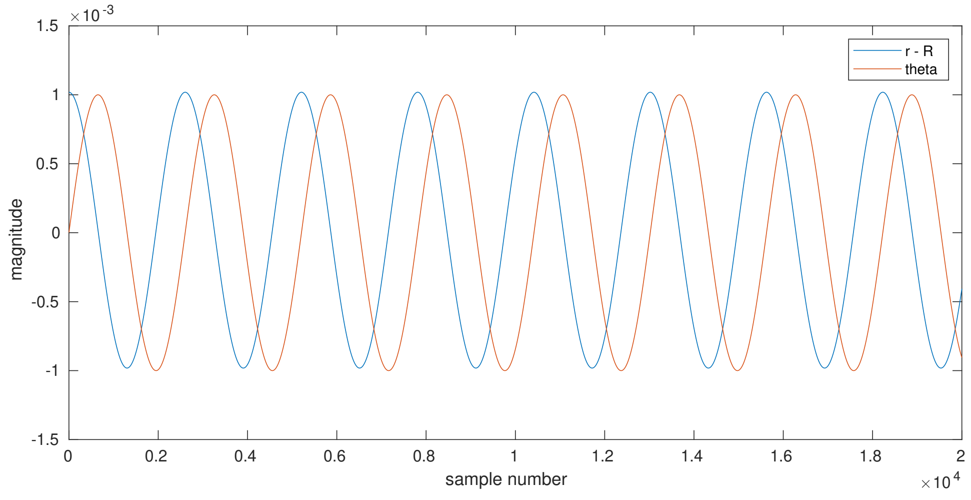
Both $r - R$ and θ move in a sinusoidal manner, approximately 90° apart from each other. When the Doppler frequency is positive, θ leads $r - R$ by 90°. When the Doppler frequency is negative, θ lags $r - R$ by 90°. This angle of separation is a function of the amplitude of the Doppler component of the receiver signal, but as the amplitude of the Doppler component approaches 0, the angle approaches 90°. Since the amplitude of the Doppler frequency is small, this is a good approximation for the system.

With this information, the sign is extracted by sending θ through a 90° delay filter and through a 90° advance filter then correlating the output of each filter with $r - R$. If the delay filter correlation is higher than the advance filter, then the sign is positive. If the advance filter correlation is higher, then the sign is negative.

Figure 4.11 shows the case where there is a carrier frequency offset on the signal of



(a) Positive Doppler



(b) Negative Doppler

Fig. 4.10: Comparison of $r - R$ and θ for both the positive and negative Doppler cases.

interest, which is the case for the constellation of $s(k)$, the signal coming out of the symbol timing synchronization block. In this case, $r - R$ is the same value, but θ now continuously increases or decreases as the entire constellation rotates. The Doppler echo symbols rotate as helices.

In this case, trying to correlate will not work very well, since θ now increases or decreases linearly as it oscillates, as shown in Figure 4.12. It is harder to distinguish θ lagging and leading $r - R$. Trying to apply the correlation does not work because of how different the

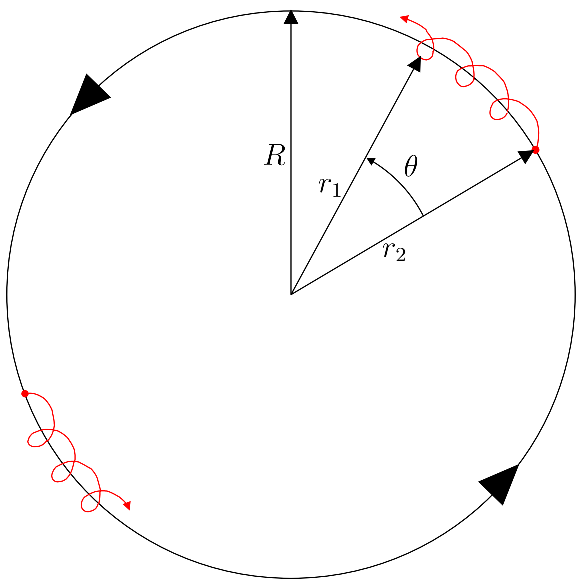


Fig. 4.11: Diagram of how Doppler frequencies rotate on a rotating signal constellation.

two values are. However, if the carrier frequency offset is removed from the signal, then θ will not increase or decrease linearly.

Removing the carrier frequency offset can either be done using the method described in Section 3.3 or by fitting a line to the plot of θ and subtracting the line. Once θ is no longer increasing or decreasing, then the correlation described previously in this section will work.

This correlation information can now correct the symmetry problem. Figure 4.13 shows a symmetric DFT spectrum of the First DEM with a Viterbi algorithm applied to only the negative frequency bins. This allows the algorithm to only follow the magnitude of the frequencies. The sign is corrected for by multiplying each frequency by either a 1 or a -1 depending on which of the correlation values are higher. This results in Figure 4.14.

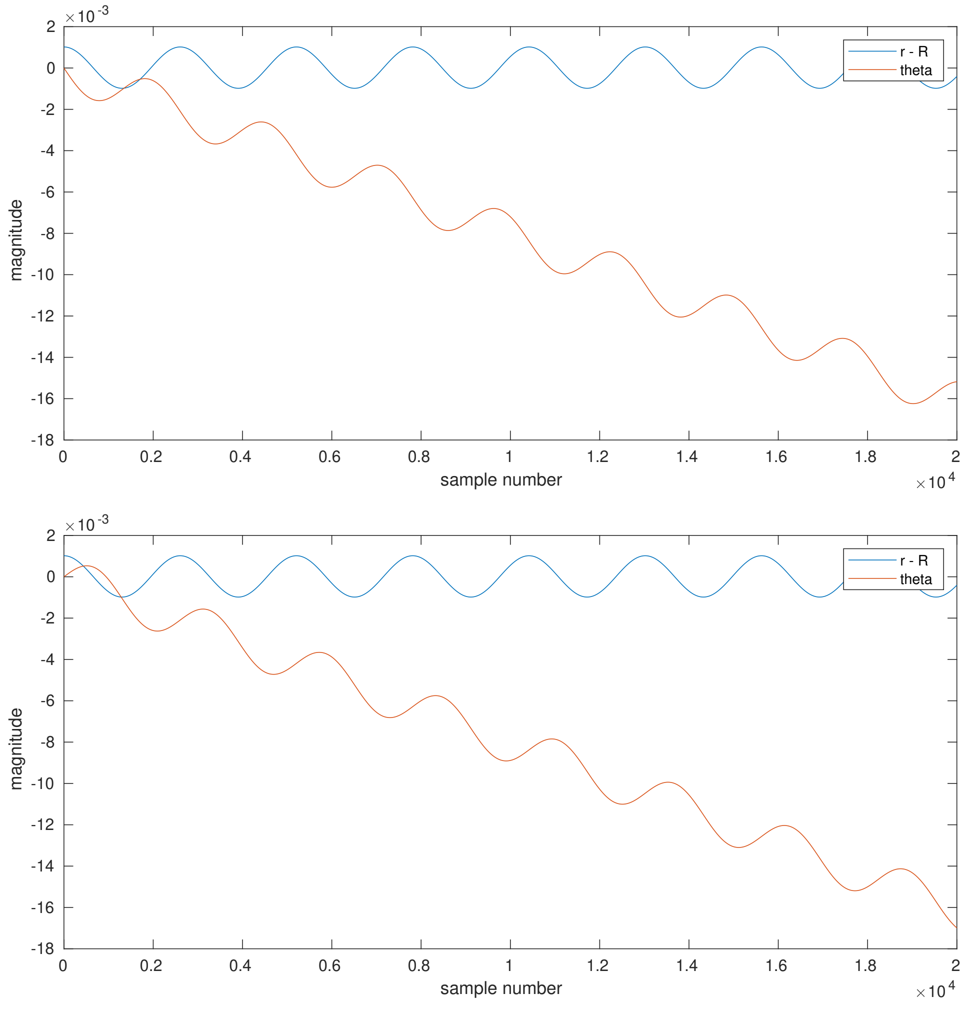


Fig. 4.12: Comparison of $r - R$ and θ with a carrier phase offset, for both the positive and negative Doppler cases.

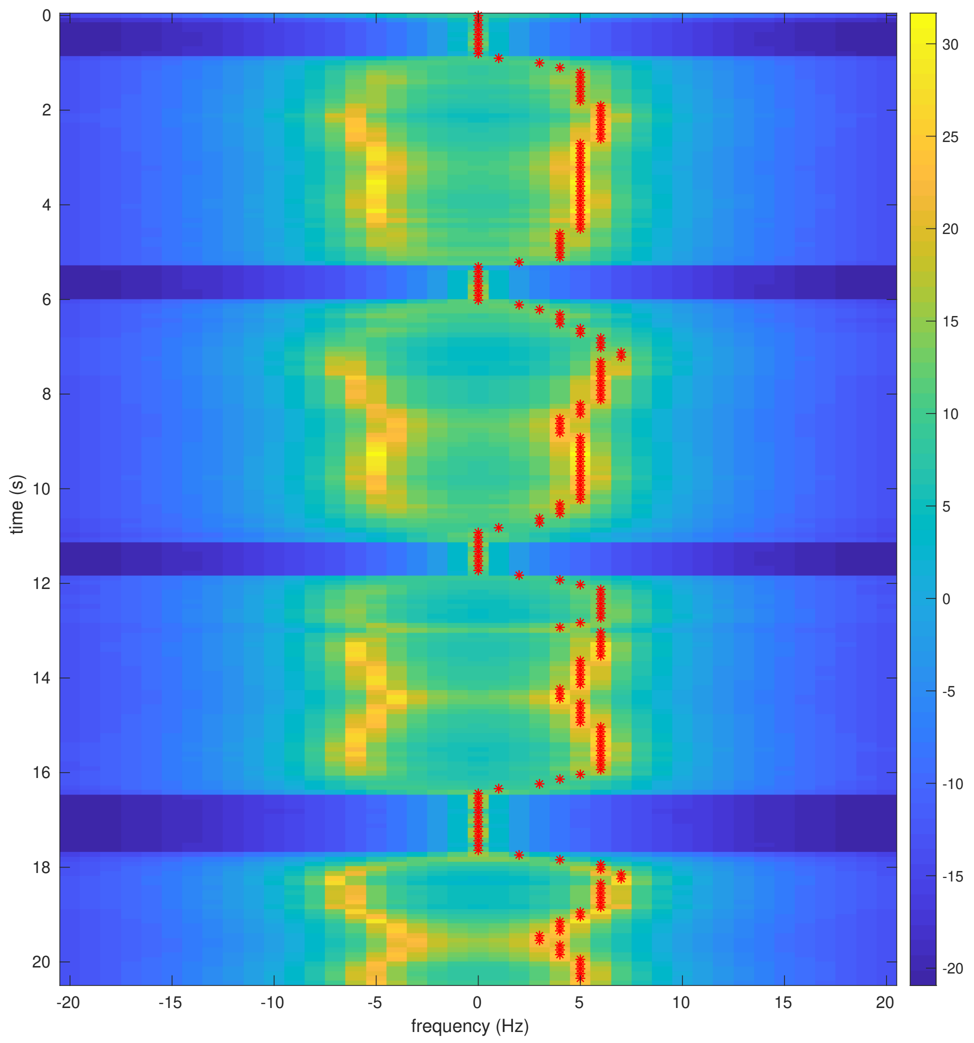


Fig. 4.13: A symmetric MUSIC spectrum of the Doppler echo signal.

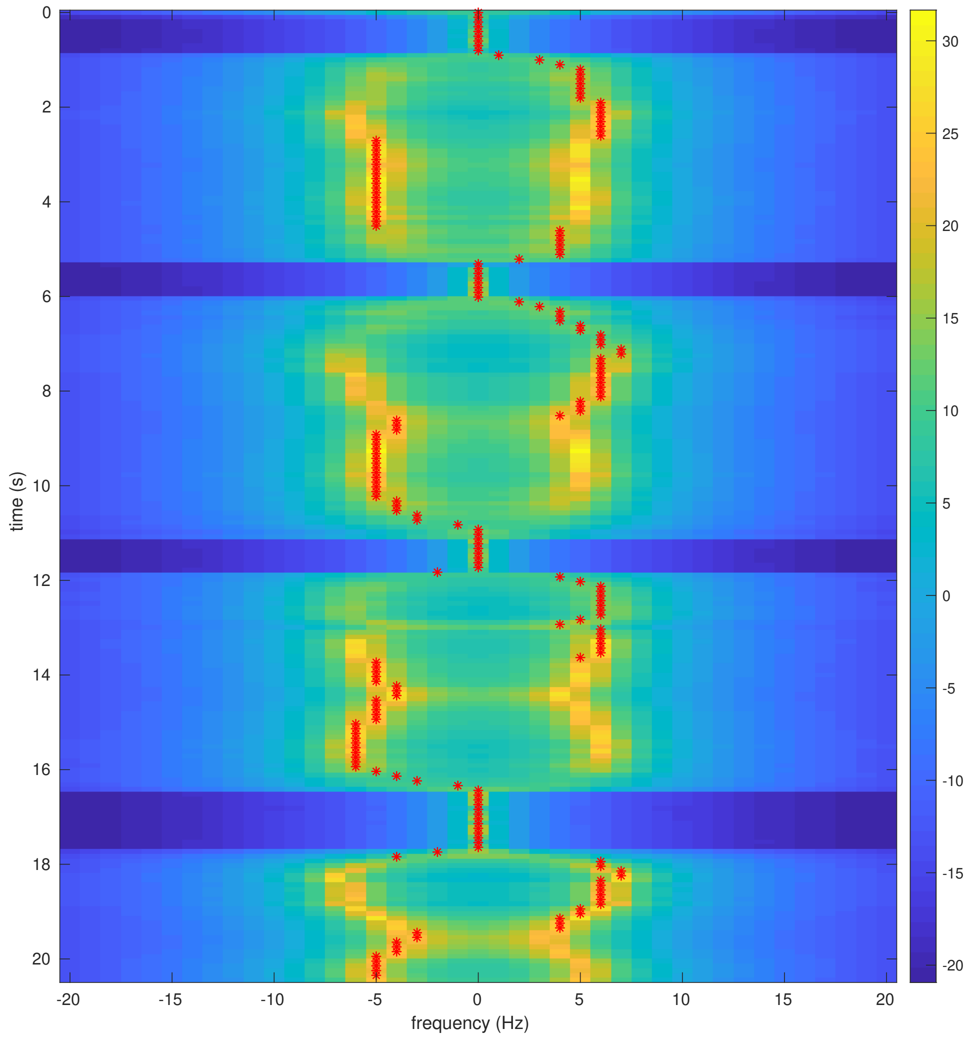


Fig. 4.14: Result of applying symmetry correction.

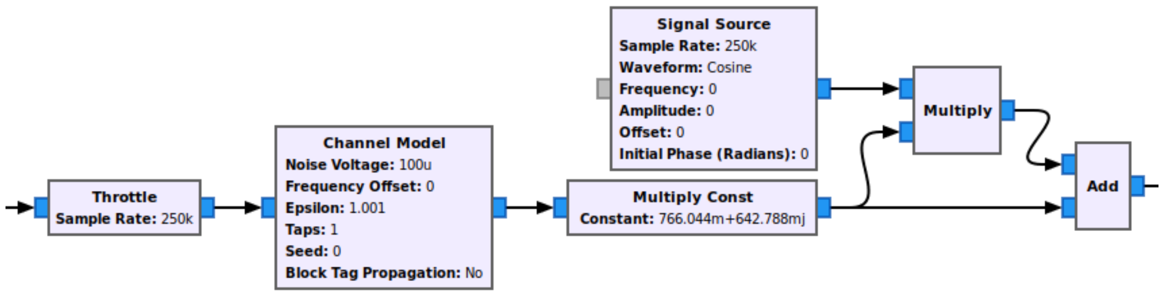
CHAPTER 5
RESULTS

This chapter describes the results of extracting the Doppler frequency path using the system of Figure 1.3. This chapter will go over the various test setups and the produced spectrograms and Doppler frequency paths. The parameters for generating the figures in this chapter are all recorded in Appendix A. Other parameters related to how the data was collected are mentioned in each specific test set section.

5.1 Simulation Results

The Doppler system simulation was done in GNU Radio. The same flowchart used in the full system, as described in Chapter 2, was used for the simulation, with one difference. Instead of the flowchart sending data into a transmitter block and then receiving the data using the receiver block, the transmitted data is sent through a simulated channel (Figure 5.1).

This simulated channel uses a Channel Model block, which models the noise, frequency offset, and timing offset of a typical channel. In addition, the simulated channel also has a Multiply Constant block to model a phase offset. Finally, the channel models the Doppler



Throttle
Sample Rate: 250k

Channel Model
Noise Voltage: 100u
Frequency Offset: 0
Epsilon: 1.001
Taps: 1
Seed: 0
Block Tag Propagation: No

Signal Source
Sample Rate: 250k
Waveform: Cosine
Frequency: 0
Amplitude: 0
Offset: 0
Initial Phase (Radians): 0

Multiply Const
Constant: 766.044m+642.788mj

Fig. 5.1: GNU Radio simulated channel with a simulated Doppler echo.

echo by modulating the signal using a signal source and then adding the resulting signal to the original signal.

This simulated channel allowed the First DEM to be tested and led to the discovery of the double frequency problem and an understanding of the limitations of the system, such as the power of the DC component compared to the power of the Doppler echo component.

Figure 5.2 shows the simulation of the double frequency problem discussed in Section 2.1.1. The two symmetric peaks in this simulation are exaggerated in height in order to see

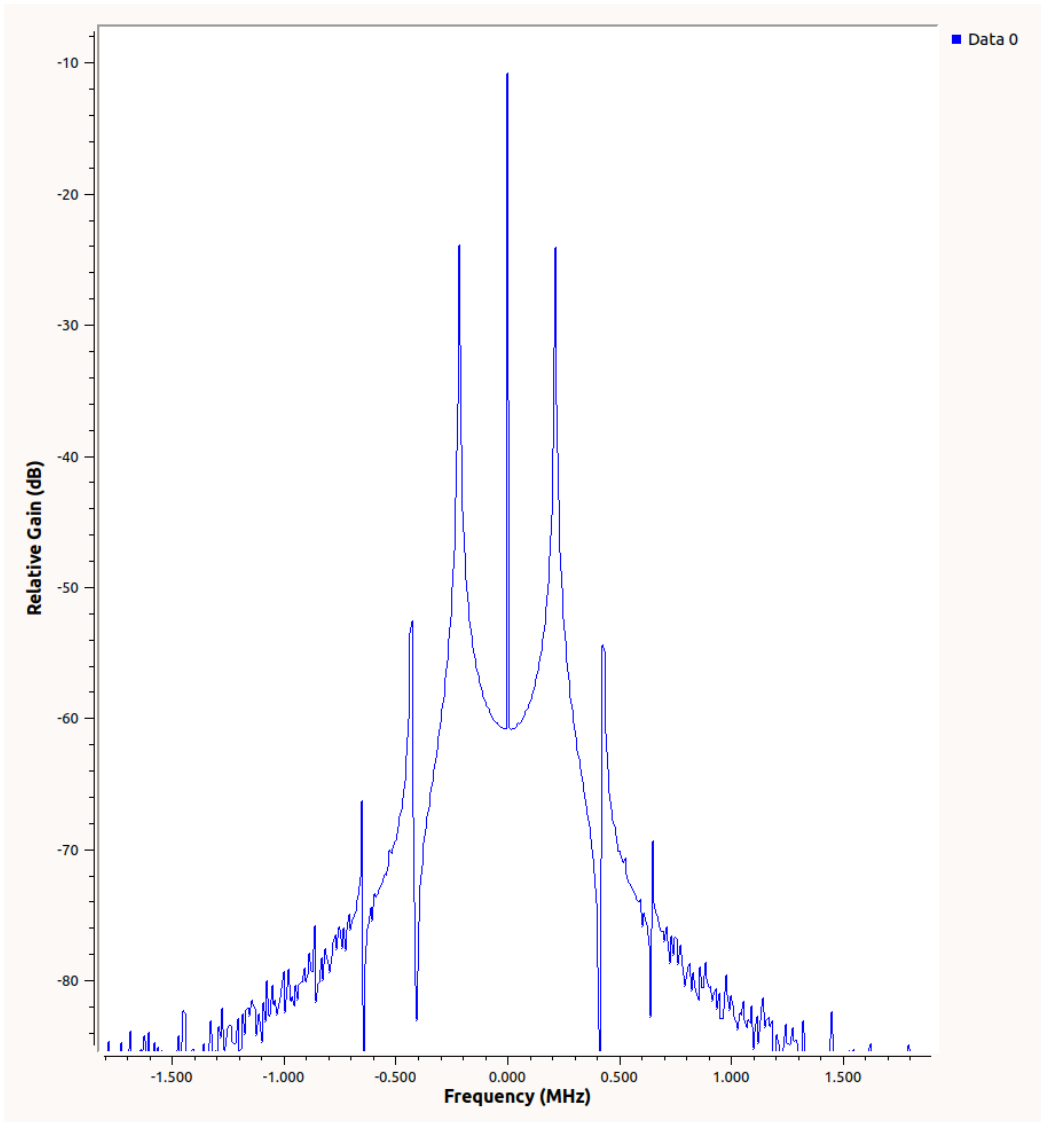


Fig. 5.2: Simulation of the double frequency issue caused by the carrier phase PLL.

the issue more clearly. The two peaks would be closer to the noise floor in a real situation.

This simulation shows how hard it is to differentiate the false peak from the real peak of the Doppler frequency. The simulation also reveals that the system creates harmonics of the Doppler frequency with lower amplitudes. These harmonics do not interfere with the Doppler extraction, but it does mean the PLL creates additional noise as it tries to track the Doppler.

5.2 Implementation Results

The Doppler extraction system was tested over a period of seven months. These tests all used the communication system described in Chapter 2, with various different configurations of transmitters, receivers, antenna placement, and number of amplifiers. The GNU Radio parameters also varied from test set to test set.

The following sections describe the setup for collecting the data for each test set. Various Doppler extraction plots are also provided to show the path the MATLAB code was able to extract from the data, with the parameters for these plots being recorded in Appendix A.

Each test set varied different parameters of the setup. Therefore, the corresponding section for each set of tests records the specific parameters of the test set. Each section describes the placement of the antennas and contains four tables. The first table describes the device parameters:

- Transmitter information
- Transmitter gain in dB
- Receiver information
- Receiver gain in dB
- Number of amplifiers used
- Sample rate of the communication system

- Interpolation and decimation rate
- Carrier frequency
- Master clock rate of the device, if configurable

The second table describes the symbol timing synchronization and pulse shape information:

- Symbol rate
- Excess bandwidth
- Matched filter length
- Timing error detector (TED)
- Expected TED gain
- Loop bandwidth
- Damping factor
- Maximum deviation
- Interpolating resampler

The third table contains the information for the carrier phase synchronization:

- B_nT_s, the loop bandwidth
- ζ, the damping factor
- K_0
- K_p

The values in the third table are used to calculate the K_1 and K_2 values using [15, pp. 684]

$$K_1 = \frac{1}{K_0K_p}\frac{\dfrac{4\zeta}{N}\left(\dfrac{B_nT_s}{\zeta+\frac{1}{4\zeta}}\right)}{1+\dfrac{2\zeta}{N}\left(\dfrac{B_nT_s}{\zeta+\frac{1}{4\zeta}}\right)+\left(\dfrac{B_nT_s}{N\left(\zeta+\frac{1}{4\zeta}\right)}\right)^2} \tag{5.1}$$

$$K_2 = \frac{1}{K_0 K_p} \frac{\frac{4}{N^2}\left(\frac{B_n T_s}{\zeta + \frac{1}{4\zeta}}\right)}{1 + \frac{2\zeta}{N}\left(\frac{B_n T_s}{\zeta + \frac{1}{4\zeta}}\right) + \left(\frac{B_n T_s}{N\left(\zeta + \frac{1}{4\zeta}\right)}\right)^2}. \tag{5.2}$$

The final table describes the different tests conducted in the particular test set. Each test in a test set differs in length and in the path of the moving target. Some tests had the moving target carrying a sheet of aluminum to improve the strength of the Doppler echo. Not all tests taken are provided as duplicate tests are not listed in these tables.

5.2.1 June 26, 2020

The first test set collected used two Heyrtz retractable antennas, one for transmitting and the other for receiving. These antennas were affixed on top of a cubicle wall and positioned a foot apart. They were both connected to a single E320 device, which acted as a transmitter and a receiver. The tests consisted of the moving target suddenly moving a short distance towards the array. The setup for these tests is recorded in Tables 5.1 – 5.3 and the tests are recorded in Table 5.4.

This first test set was an initial inquiry into how strong the Doppler echo would be. Most of the tests conducted were short and were mainly focused on a sudden movement forward. Other tests were conducted but were less useful than these. As these tests were some of the first conducted, the tests only collected data from the First DEM.

Table 5.1: Device Parameters, June 26, 2020

Transmitter	E320 #41, RF A, Tx/Rx
Transmitter gain	89.75 dB
Receiver	E320 #41, RF B, Tx/Rx
Receiver gain	36 dB
Number of amplifiers	0
Sample rate	31250 Samples/s
Interpolation/Decimation rate	1
Carrier frequency	905 MHz
Master clock rate	500 kHz

Table 5.2: Symbol Timing Synchronization Parameters, June 26, 2020

Samples per symbol	4
Excess bandwidth	0.5
Matched filter length	161
TED	y[n]y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.1
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.3: Carrier phase synchronization parameters, June 26, 2020

B_nT_s	90 μ
ζ	0.707
K_p	0.5
K_0	1

The PLL problem did not affect this data because there was no need to perform carrier phase synchronization. Only one device was used for both transmitting and receiving data, which means the same clock was used for modulating and demodulating the communication signal. The B_nT_s value was turned down to only adjust for a phase offset and not a frequency offset, which means the Doppler frequency was filtered out by the loop filter.

Figures 5.3 – 5.5 show the resulting DFT spectrograms with a path extracted by the

Table 5.4: Tests, June 26, 2020

Test	Metal	Description
1	Yes	Moving towards the antennas at an angle perpendicular to the antennas, slower pace
2	Yes	Moving towards the antennas at an angle perpendicular to the antennas, normal pace
3	Yes	Moving towards the antennas at an angle perpendicular to the antennas, faster pace
4	No	Moving towards the antennas at an angle perpendicular to the antennas, slower pace
5	No	Moving towards the antennas at an angle perpendicular to the antennas, normal pace
6	No	Moving towards the antennas at an angle perpendicular to the antennas, faster pace

Viterbi algorithm for tests 1, 5, and 6. These tests had the best results as they all showed a perturbation of the Doppler frequencies due to the sudden movement forward. Test 1 was the only one of these tests to use the sheet metal to improve the reflection strength.

An important observation over all of these spectrograms is the two bands of energy on either side of the DC value. These energy bands are remnants of the strong DC component. The DC was removed here by taking a data frame, estimating the mean of the data frame, and then subtracting the mean from the data frame before computing the spectrum for the frame. Then, the DC for the spectrum is scaled up to the mean of the spectrum. This results in the spectrogram having better contrast.

The energy bands are not as strong as the Doppler, but they do make it hard to track the Doppler path when the magnitude of the Doppler frequency drops below 4 Hz. This makes subtle changes in Doppler at lower frequencies harder to track.

It is important to note how spread out the Doppler is. This is a result of the DFT, as the Doppler frequency could change by 1 – 2 Hz over the data frame length, which is 0.2 seconds. This means the resolution of the spectrum is lower, as mentioned in Section 3.1.

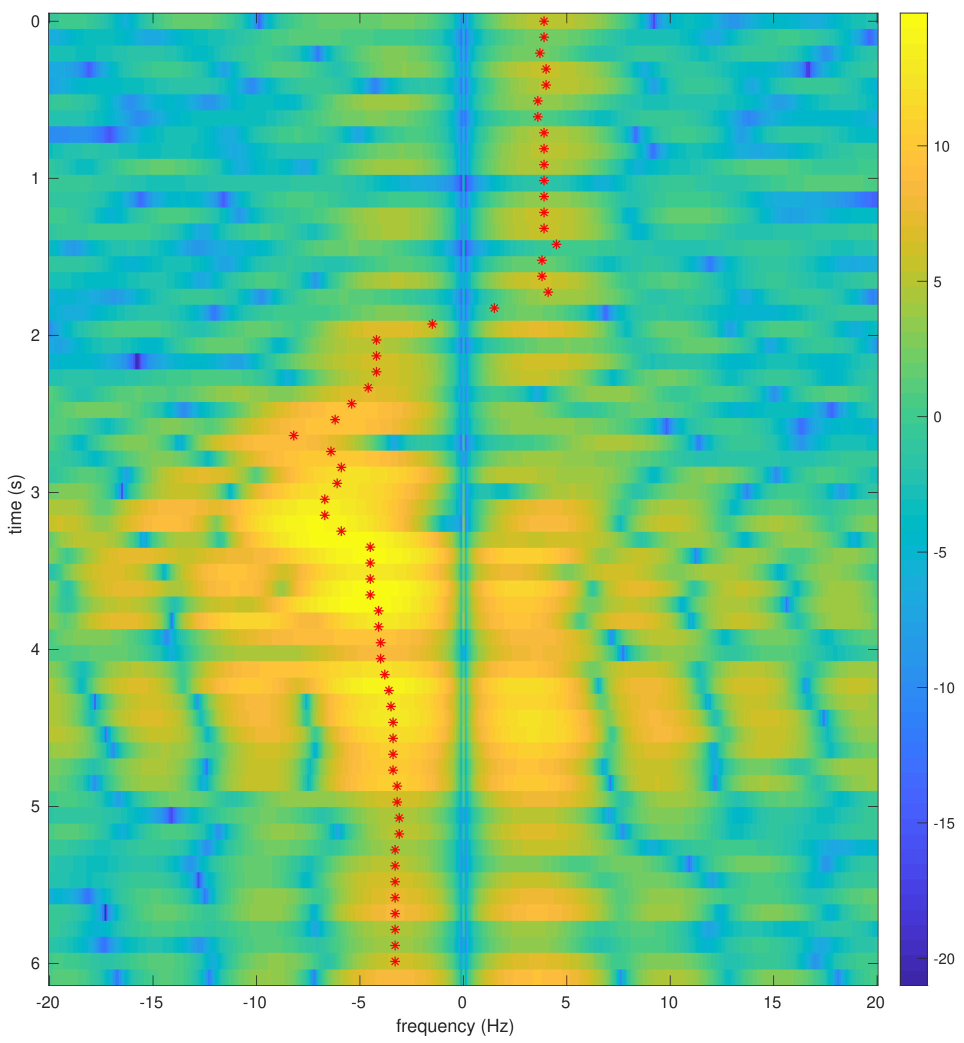


Fig. 5.3: Test 1 extracted path over a DFT spectrogram.

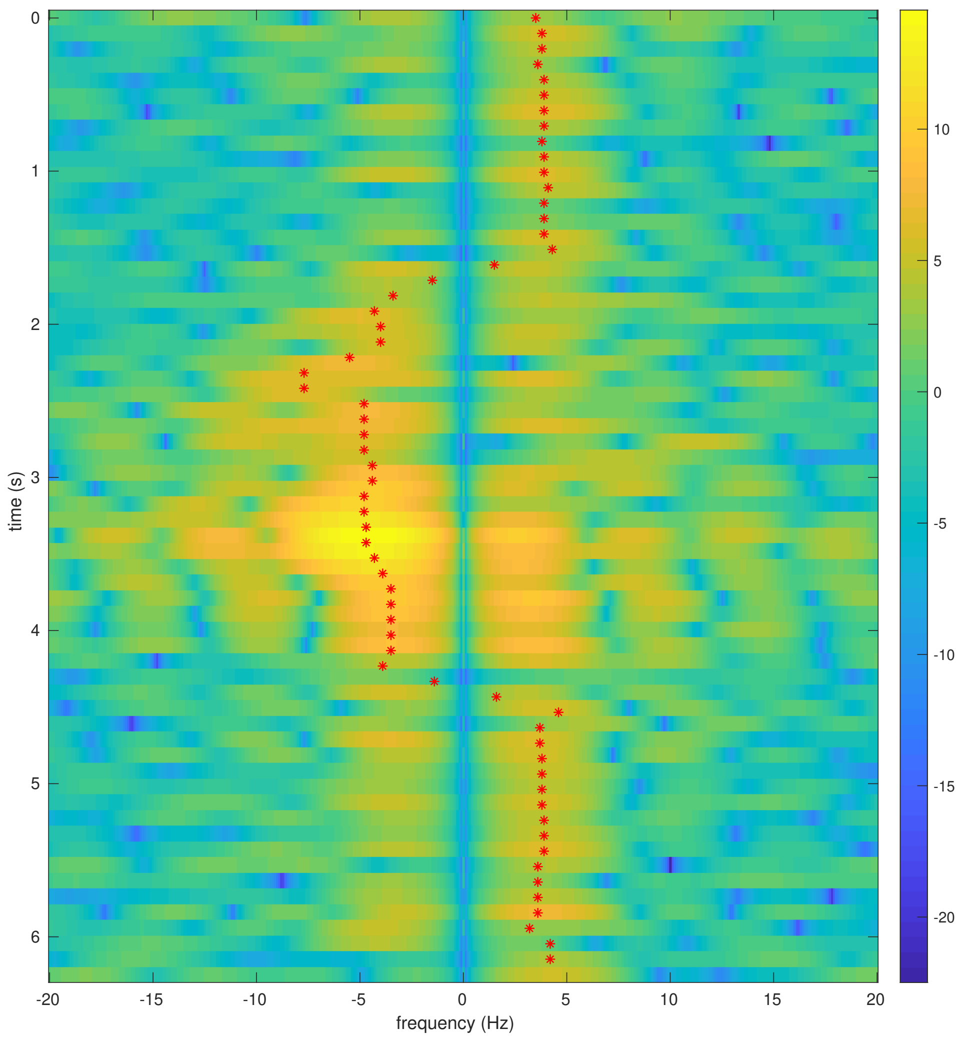


Fig. 5.4: Test 5 extracted path over a DFT spectrogram.

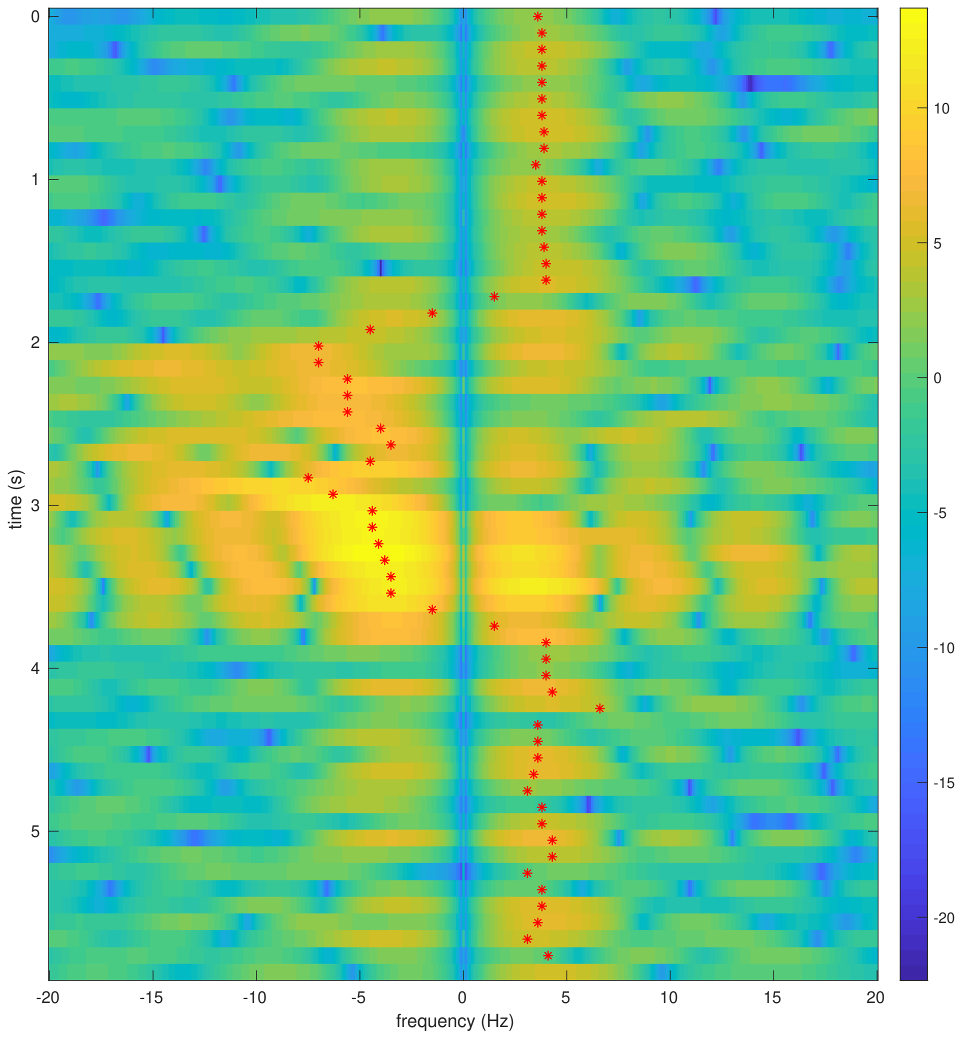


Fig. 5.5: Test 6 extracted path over a DFT spectrogram.

5.2.2 July 13, 2020

This test set has the same setup as the previous one. Two Heyrtz antennas were attached on top of a cubicle wall, a foot apart from each other, and were connected to a single E320 device. This test set introduced a new type of test, which allowed the Doppler echo path to become more apparent. The new test type involved having the moving target continuously move towards and away from the antenna array. The setup for these tests is recorded in Tables 5.5 – 5.7 and the tests are recorded in Table 5.8.

The objective of these tests was to have a continuous path of Doppler frequencies to avoid the problem of the Doppler sitting at DC. Figures 5.6 to 5.9 show the results of the

Table 5.5: Device Parameters, July 13, 2020

Transmitter	E320 #41, RFA, Tx/Rx
Transmitter gain	89.75 dB
Receiver	E320 #41, RFB, Tx/Rx
Receiver gain	36 dB
Number of amplifiers	0
Sample rate	31250 Samples/s
Interpolation/Decimation rate	1
Carrier frequency	905 MHz
Master clock rate	500 kHz

Table 5.6: Symbol Timing Synchronization Parameters, July 13, 2020

Samples per symbol	4
Excess bandwidth	0.5
Matched filter length	161
TED	y[n]y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.1
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.7: Carrier phase synchronization parameters, July 13, 2020

B_nT_s	90 μ
ζ	0.707
K_p	0.5
K_0	1

Table 5.8: Tests, July 13, 2020

Test	Metal	Description
1	Yes	Moving towards and away from the antenna array repeatedly
2	Yes	Moving towards and away from the antenna array repeatedly
3	No	Moving towards and away from the antenna array repeatedly
4	No	Moving towards and away from the antenna array repeatedly

in and out motion on the Doppler frequency path. The Doppler frequency path oscillates in all of these figures and shows the ability of the Viterbi algorithm to find this path in the noise more easily than the previous test sets.

These tests were the first to compare the DFT and the MUSIC algorithm. The MUSIC algorithm overall does a better job at extracting the signal than the DFT. The spectrogram created by MUSIC has less spread due to the increase in resolution. However, MUSIC is computationally more complex, especially since $P = 50$ and $M = 200$ for both of the MUSIC spectrograms.

Once again, the tests with the metal are easier to extract the Doppler path from than the tests without the metal. This is more clear in these figures than those in the last test set as the MUSIC spectrogram becomes noisier in Figure 5.9 than in Figure 5.8.

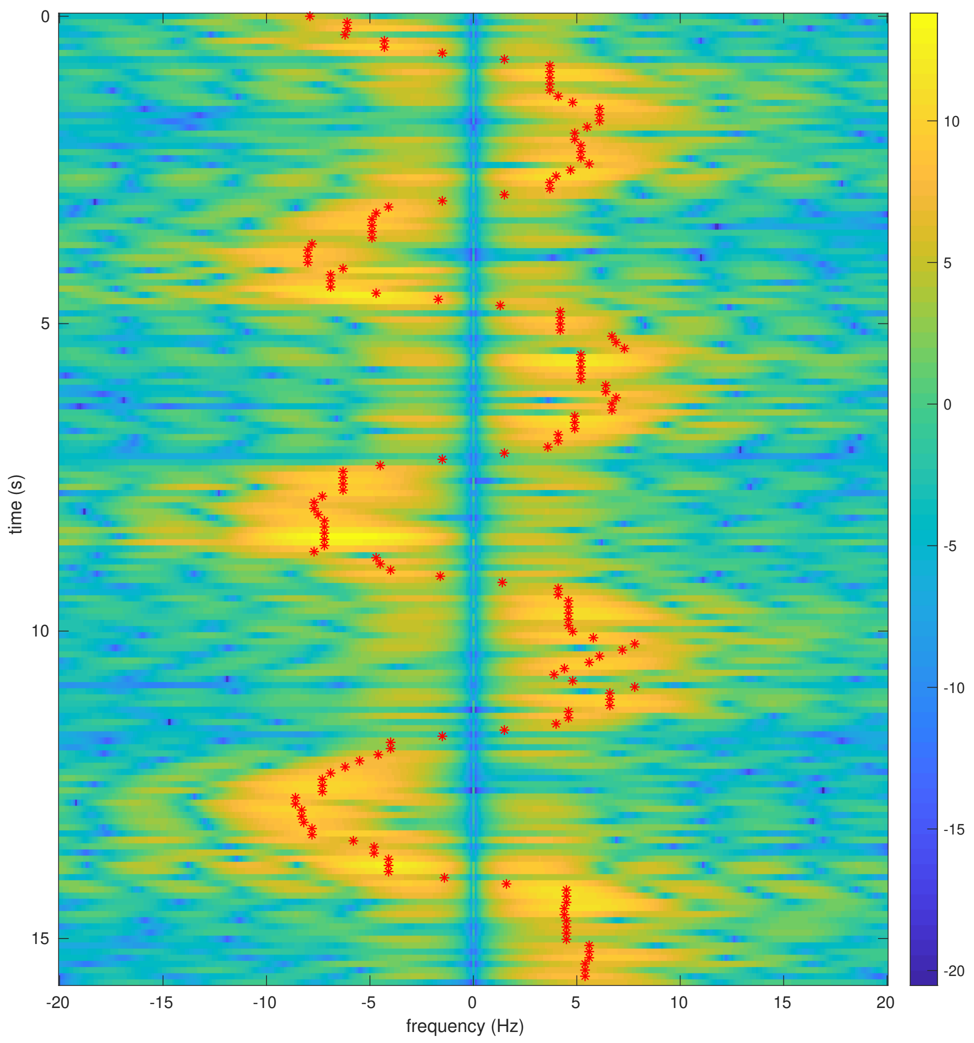


Fig. 5.6: Test 1 extracted path over a DFT spectrogram.

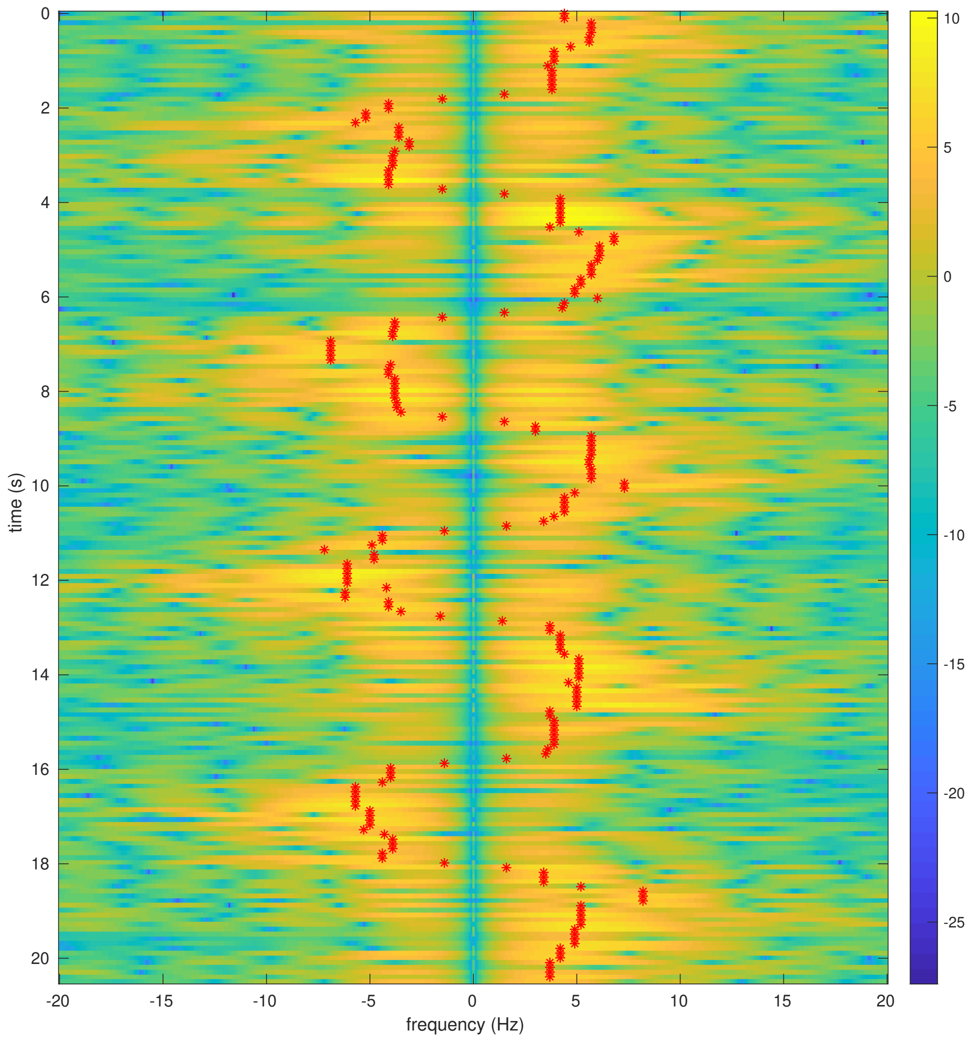


Fig. 5.7: Test 3 extracted path over a DFT spectrogram.

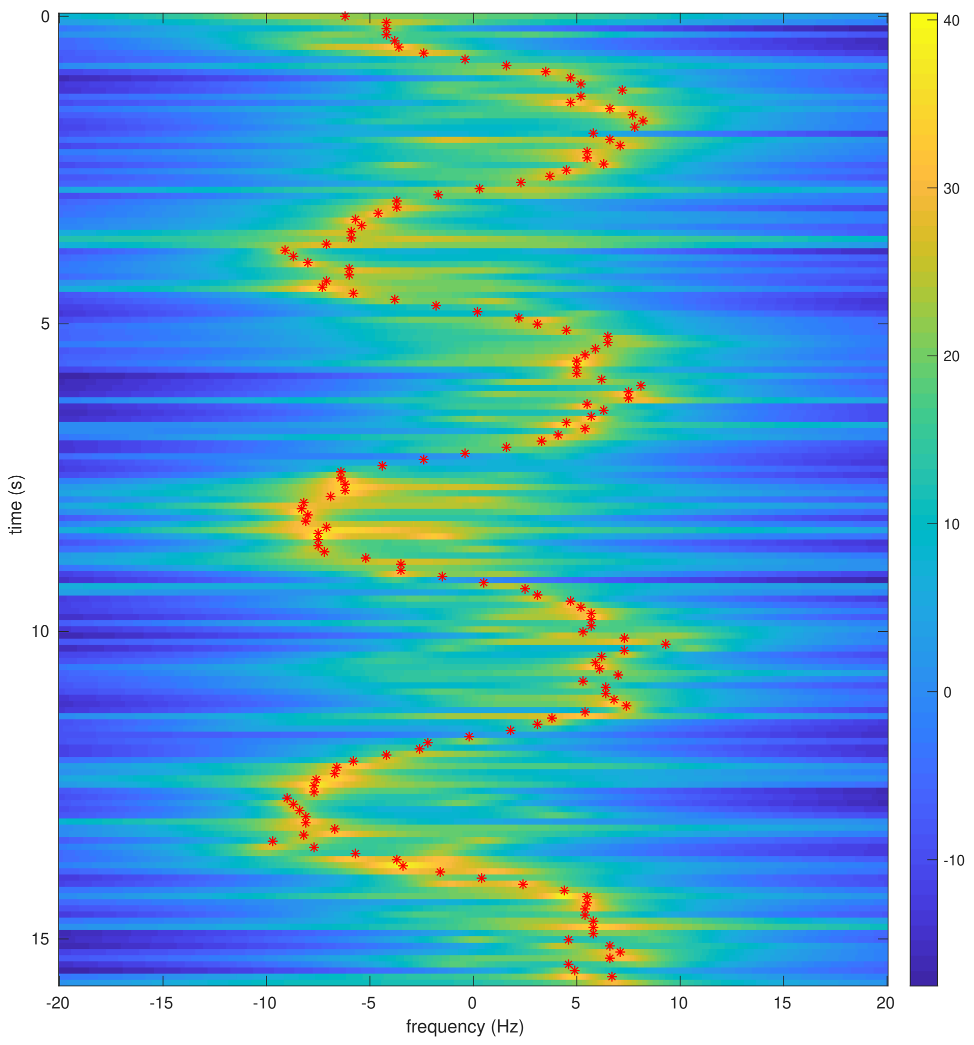


Fig. 5.8: Test 1 extracted path over a MUSIC spectrogram.

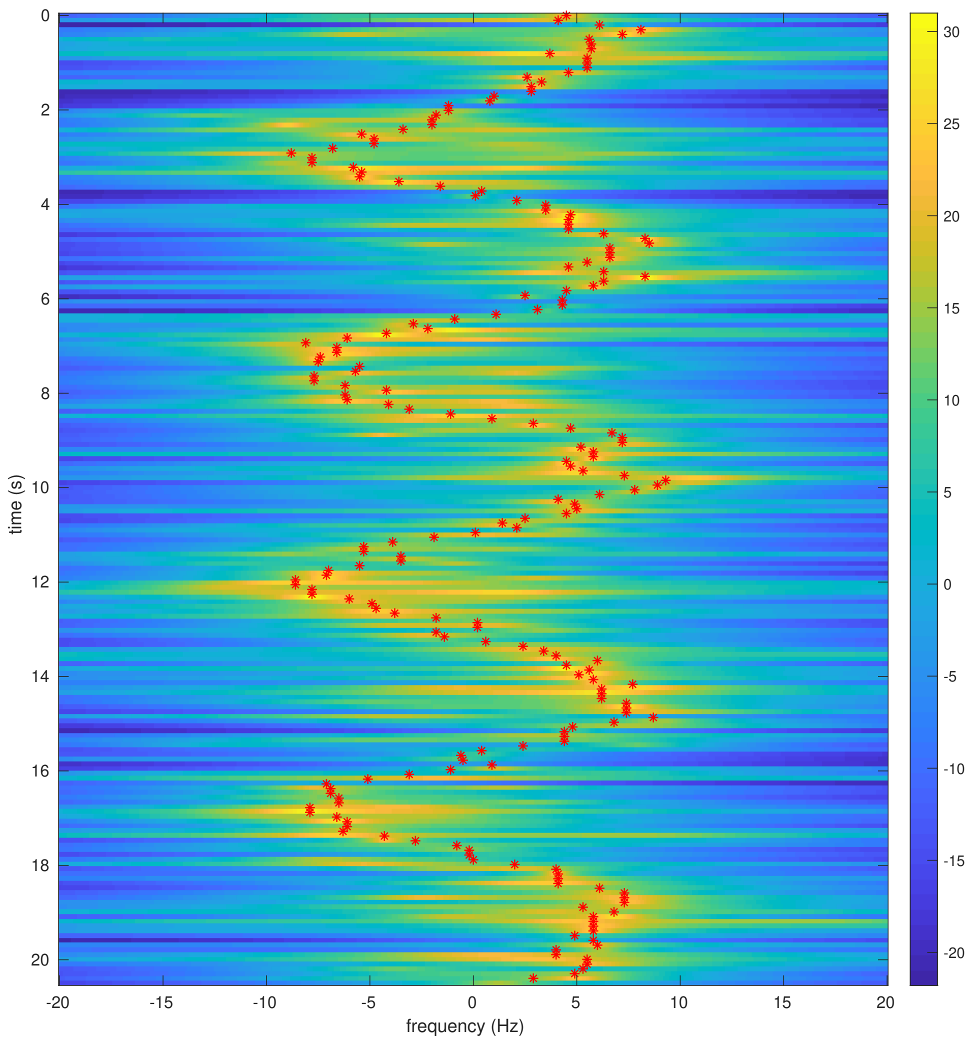


Fig. 5.9: Test 3 extracted path over a MUSIC spectrogram.

5.2.3 July 15, 2020

These tests once again used two Heyrtz antennas attached to the top of a cubicle a foot apart from each other. The setup for these tests was nearly identical to the previous tests, except for the loop bandwidth being higher and the use of two E320s. The setup for these tests is recorded in Tables 5.9 – 5.11 and the tests are recorded in Table 5.12.

These tests were the first tests using both Doppler extraction methods. The purpose of these tests was to see if the Second DEM was able to more accurately extract the Doppler than the First DEM when there was a carrier frequency offset.

Figure 5.10 shows how the First DEM fails when there is a carrier frequency offset. The spectrogram becomes symmetric and makes it difficult for the Viterbi algorithm to find the correct path through the spectrogram. There is also additional noise besides the symmetric Doppler peak. This suggests the harmonics seen in the simulation have more effects than the simulation suggested.

Table 5.9: Device Parameters, July 15, 2020

Transmitter	E320 #41, RFA, Tx/Rx
Transmitter gain	89.75 dB
Receiver	E320 #40, RFB, Tx/Rx
Receiver gain	62 dB
Number of amplifiers	0
Sample rate	31250 Samples/s
Interpolation/Decimation rate	1
Carrier frequency	905 MHz
Master clock rate	500 kHz

Table 5.10: Symbol Timing Synchronization Parameters, July 15, 2020

Samples per symbol	4
Excess bandwidth	0.5
Matched filter length	161
TED	y[n]y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.1
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.11: Carrier phase synchronization parameters, July 15, 2020

$B_n T_s$	90 m
ζ	0.707
K_p	0.5
K_0	1

Table 5.12: Tests, July 15, 2020

Test	Metal	Description
1	Yes	Moving towards and away from the antenna array repeatedly, transmitter and receiver tied to the same clock
2	No	Moving towards and away from the antenna array repeatedly, transmitter and receiver tied to the same clock
3	Yes	Moving towards and away from the antenna array repeatedly, transmitter and receiver on different clocks
4	No	Moving towards and away from the antenna array repeatedly, transmitter and receiver on different clocks

The Second DEM is able to clean up this spectrogram and find an appropriate Doppler path as seen in Figure 5.11. The estimation of the carrier frequency and subsequent demodulation of the signal are able to remove the carrier frequency offset well enough to extract the path from the spectrogram with the Viterbi algorithm.

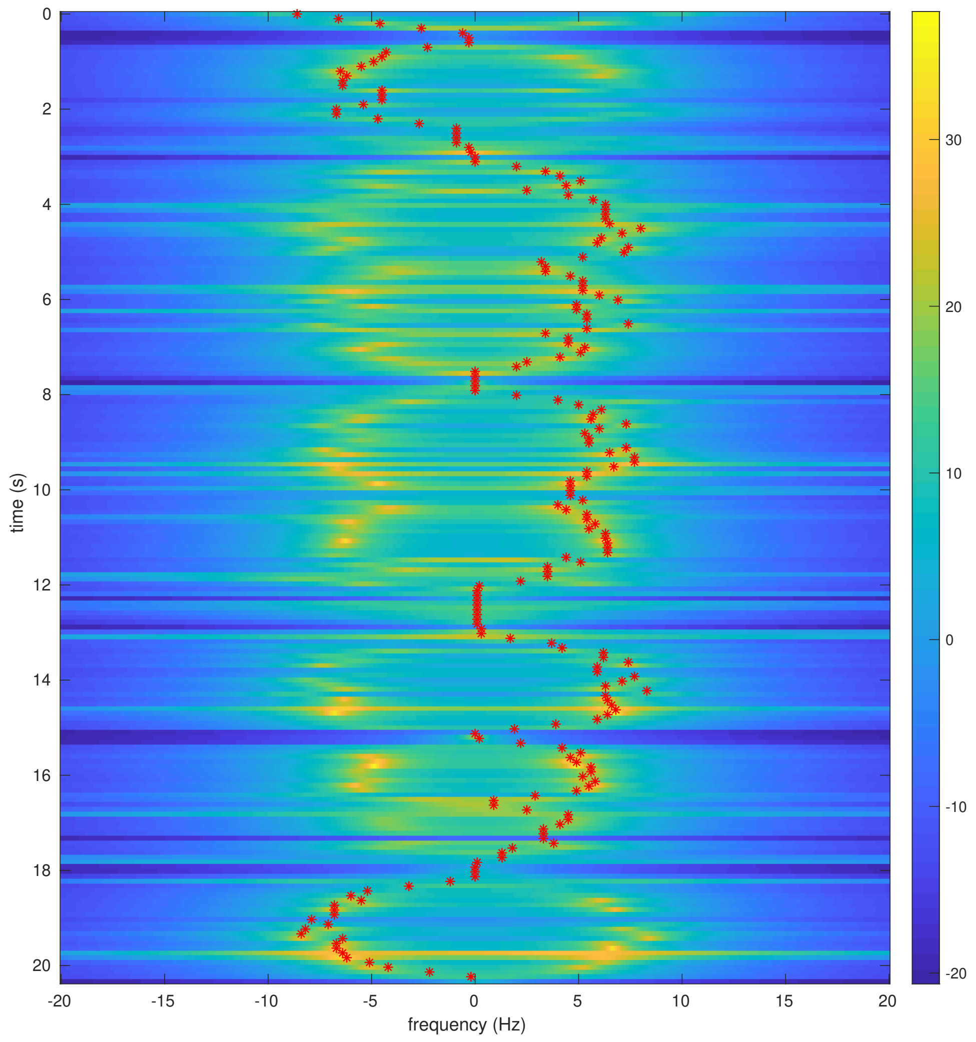


Fig. 5.10: Test 3 using the First DEM, with the optimal path plotted on the MUSIC spectrum.

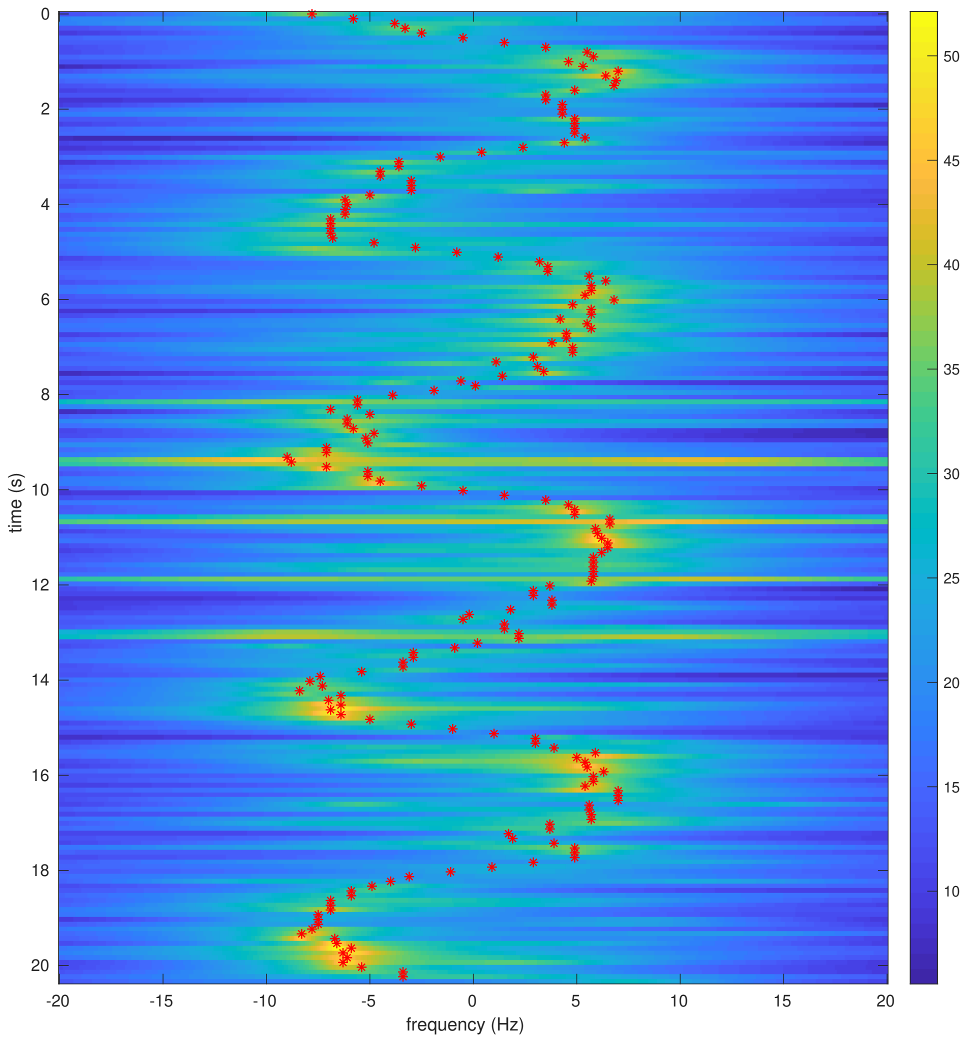


Fig. 5.11: Test 3 using the Second DEM, with the optimal path plotted on the MUSIC spectrum.

5.2.4 July 21, 2020

These tests used two Heyrtz antennas attached to the top of a cubicle a foot apart from each other. These tests were similar to the previous test sets but used different devices for the transmitter and receiver. A glitch in E320 devices caused extra phase noise. This extra phase noise made it difficult to extract the Doppler without the metal, so other devices were explored as alternatives. This test set investigated using the B200 mini and RTL-SDR devices. The setup for these tests is recorded in Tables 5.13 – 5.15 and the tests are recorded in Table 5.16.

The RTL-SDR dongle is not an optimal device for collecting the Doppler data, as it has a poor clock. A poor clock results in the carrier frequency offset becoming noisier and harder to account for in the carrier phase removal. Figure 5.12 shows the system was still able to extract a path for the Doppler despite the extra noise in Test 1, the test with metal. However, without the metal, the Viterbi algorithm lost the path partway through the test

Table 5.13: Device Parameters, July 21, 2020

Transmitter	B200 mini
Transmitter gain	75 dB
Receiver	RTL-SDR Dongle
Receiver gain	25 dB
Number of amplifiers	0
Sample rate	62500 Samples/s
Interpolation/Decimation rate	1
Carrier frequency	905 MHz
Master clock rate	n/a

Table 5.14: Symbol Timing Synchronization Parameters, July 21, 2020

Samples per symbol	8
Excess bandwidth	0.5
Matched filter length	161
TED	y[n]y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.1
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.15: Carrier phase synchronization parameters, July 21, 2020

B_nT_s	90 m
ζ	0.707
K_p	0.5
K_0	1

Table 5.16: Tests, July 21, 2020

Test	Metal	Description
1	Yes	Moving towards and away from the antenna array repeatedly
2	Yes	Moving towards and away from the antenna array repeatedly
3	No	Moving towards and away from the antenna array repeatedly
4	No	Moving towards and away from the antenna array repeatedly

as Figure 5.13 shows.

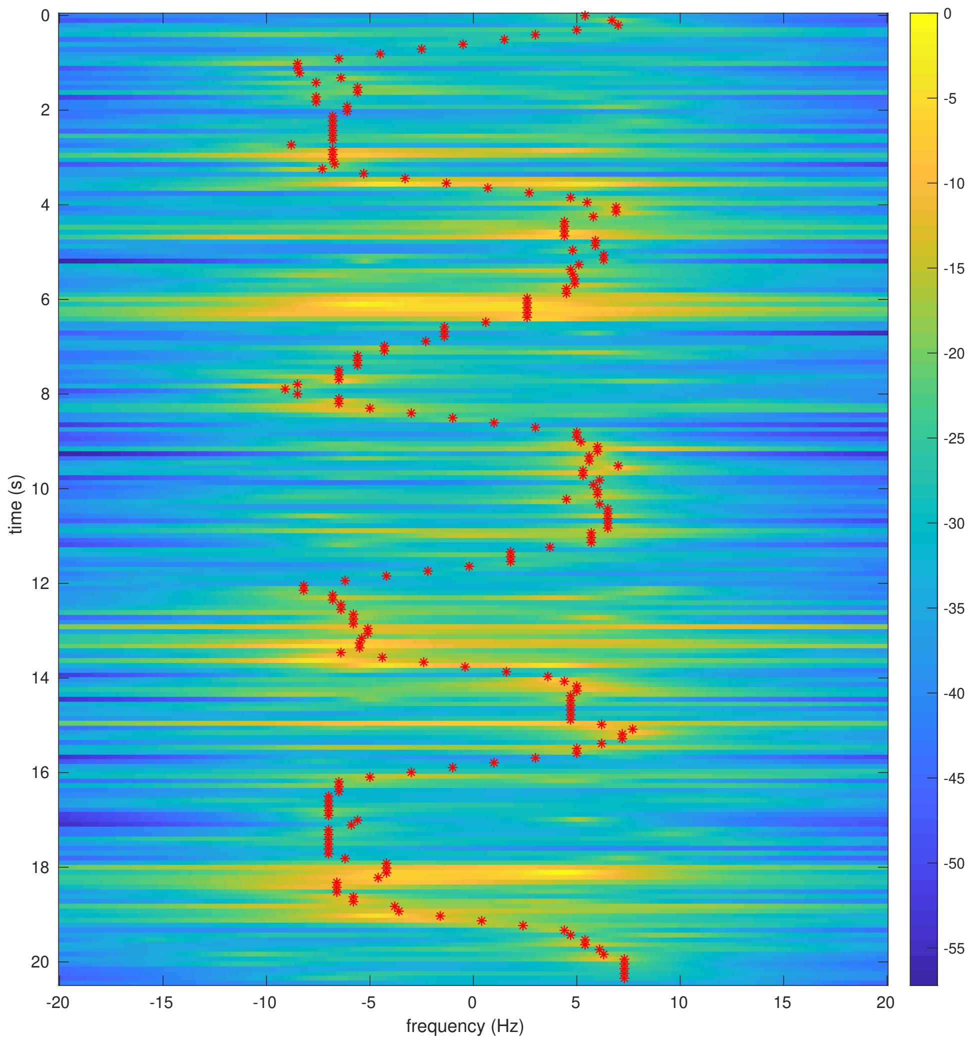


Fig. 5.12: Test 1 with optimal path plotted on MUSIC spectrum.

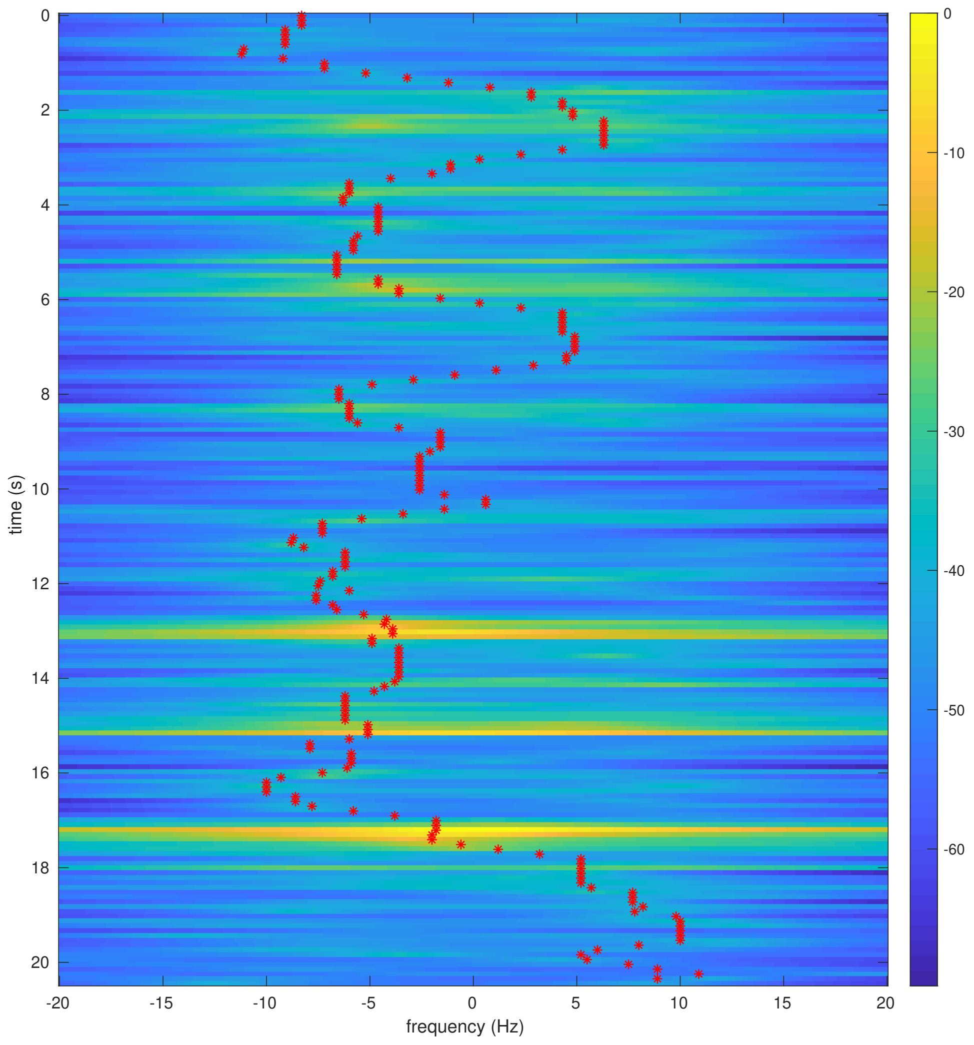


Fig. 5.13: Test 3 with optimal path plotted on MUSIC spectrum.

5.2.5 August 10, 2020

These tests used two Heyrtz antennas attached to the top of a cubicle a foot apart from each other. This test set investigated using the B200 mini and B200 devices. The setup for these tests is recorded in Tables 5.17 – 5.19 and the tests are recorded in Table 5.20.

The B200 device has a GPSDO clock on it, causing the carrier frequency offset to be less noisy than the carrier frequency offset for the RTL-SDR devices in the previous test set. This led to the ability to extract the Doppler frequency from the signal both with and without the metal. Figure 5.14 shows the Doppler extraction with the metal and Figure 5.15 shows the Doppler extraction without the metal.

Table 5.17: Device Parameters, August 10, 2020

Transmitter	B200 mini
Transmitter gain	80 dB
Receiver	B200
Receiver gain	75 dB
Number of amplifiers	0
Sample rate	62500 Samples/s
Interpolation/Decimation rate	8
Carrier frequency	905 MHz
Master clock rate	n/a

Table 5.18: Symbol Timing Synchronization Parameters, August 10, 2020

Samples per symbol	8
Excess bandwidth	0.5
Matched filter length	161
TED	y[n]y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.02
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.19: Carrier phase synchronization parameters, August 10, 2020

B_nT_s	$9\ m$
ζ	0.707
K_p	0.5
K_0	1

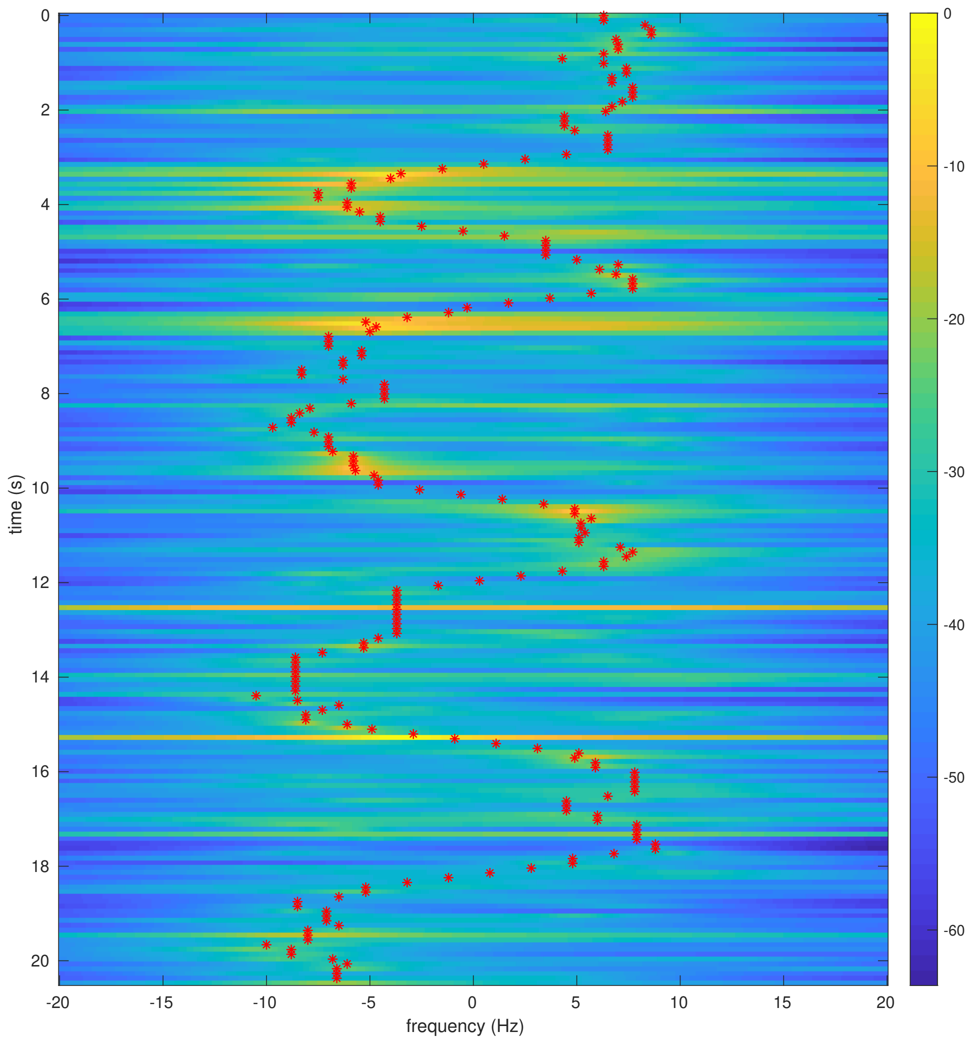


Fig. 5.14: Test 1 with the optimal path plotted on MUSIC spectrum.

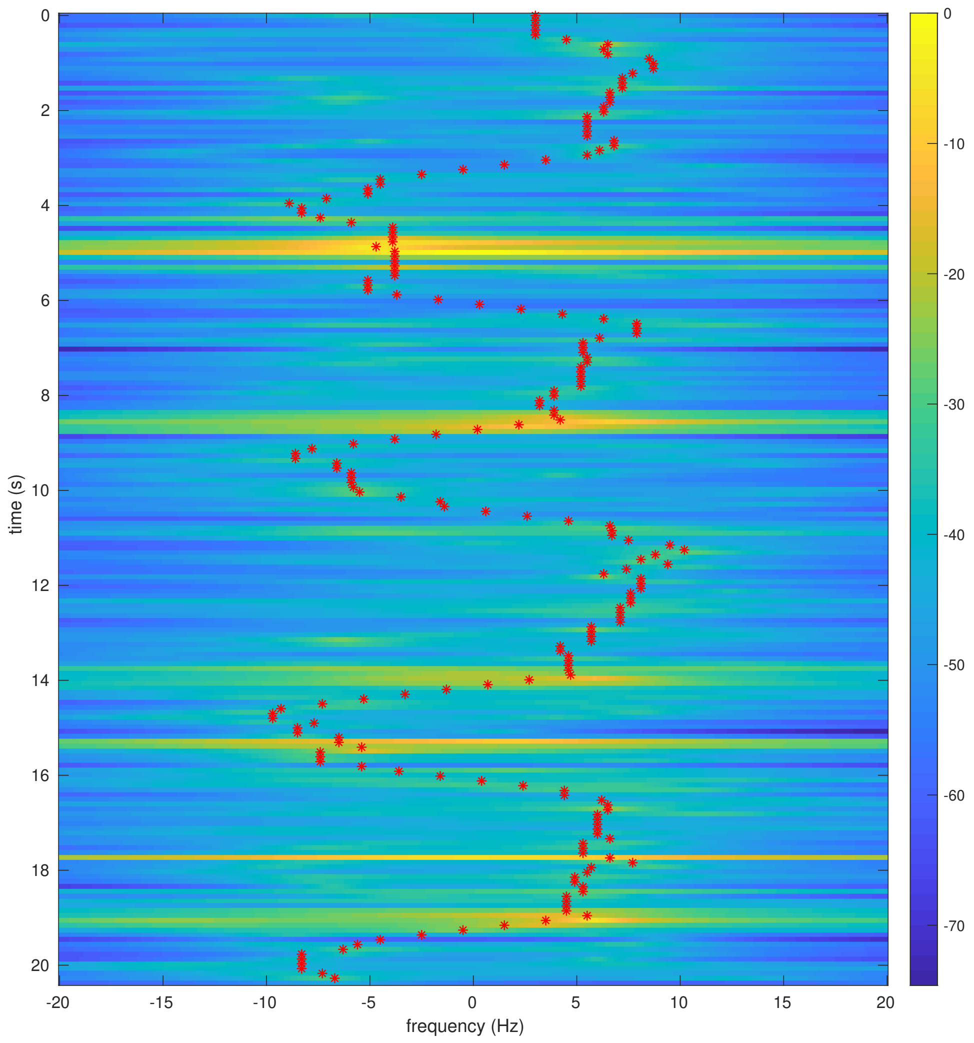


Fig. 5.15: Test 2 with the optimal path plotted on MUSIC spectrum.

Table 5.20: Tests, August 10, 2020

Test	Metal	Description
1	Yes	Moving towards and away from the antenna array repeatedly
2	No	Moving towards and away from the antenna array repeatedly
3	Yes	Moving toward array suddenly
4	No	Moving toward array suddenly
5	Yes	Walking in a clockwise circle
6	No	Walking in a clockwise circle
7	Yes	Walking in and out of the room in a counter clockwise direction
8	No	Walking in and out of the room in a counter clockwise direction

5.2.6 Simulated Doppler Paths

The rest of the test sets recorded data using the SW, NW, SC, and NC hall antennas, which were positioned as shown in Figure 2.6. These antennas are HyperLink antennas, which were better suited for the transmission frequency of the communication signal.

The correct path for these tests is harder to determine due to the geometry of the antennas being more complex than the previous setup. Therefore, a simulation program was written in order to simulate what the Doppler path might look like.

All of the following test sets have one test in common. This test is when the moving target moves in the southern hall between the SC and SW antennas. Therefore, this test was simulated to compare with the actual Doppler frequency paths.

The simulated data was merely used to get a general idea of what the path should look like. It is not a direct comparison. Human movement is hard to simulate precisely in a computer model, and, therefore, the simulated Doppler path is somewhat unrepresentative of the real Doppler path. However, the general path of these simulations should be close to the actual recorded paths. Figures 5.16 – 5.19 show the simulated paths.

Some of the tests move the position of the transmitter. These tests can still be compared to this simulated data, but caution should be used when using such comparisons as the placement of the transmitter can easily change the Doppler path at each antenna.

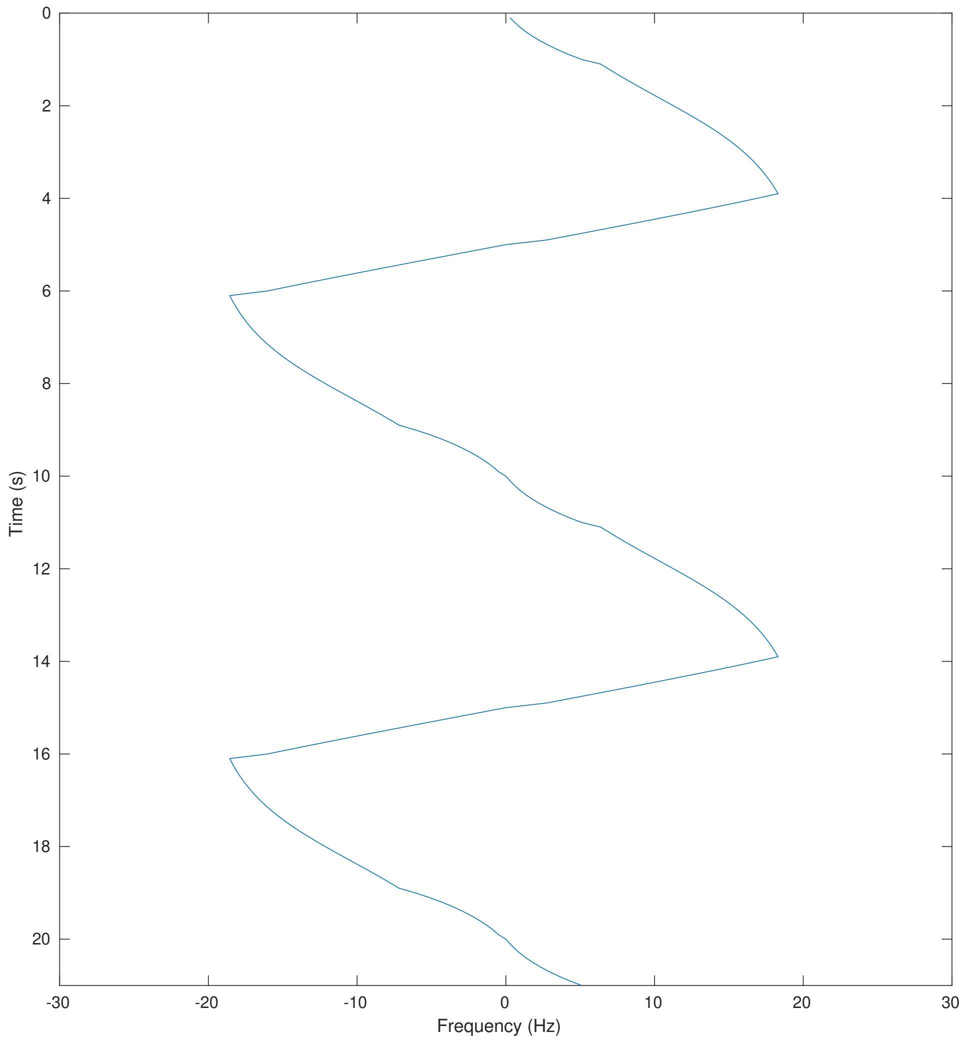


Fig. 5.16: Doppler frequency simulated at the SC antenna.

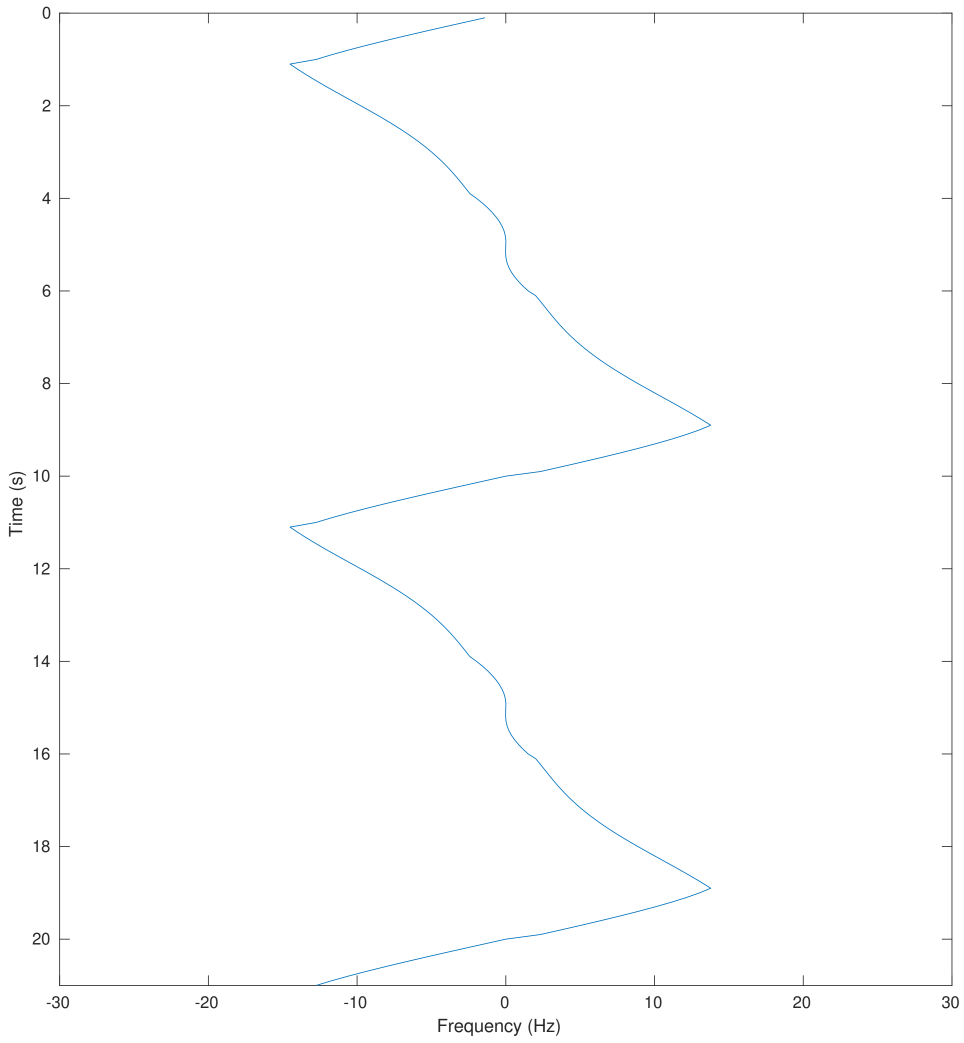


Fig. 5.17: Doppler frequency simulated at the SW antenna.

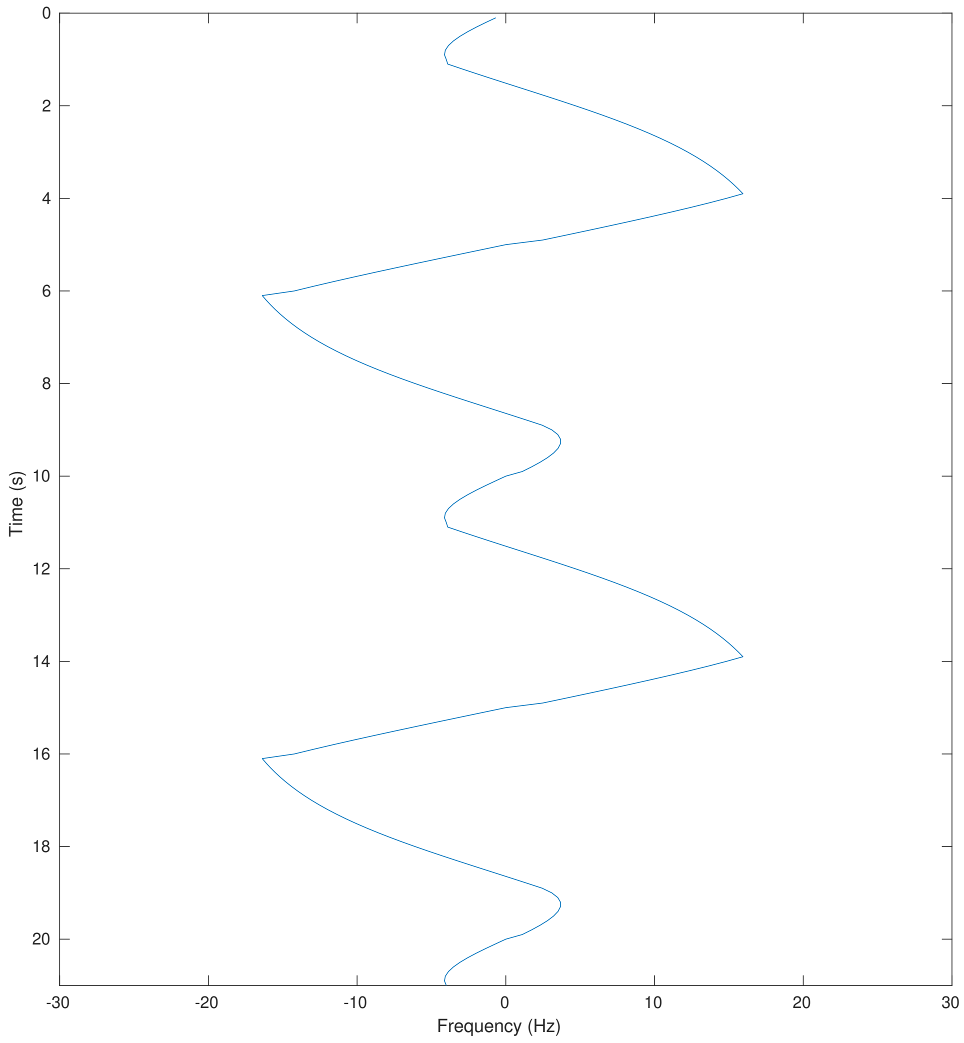


Fig. 5.18: Doppler frequency simulated at the NC antenna.

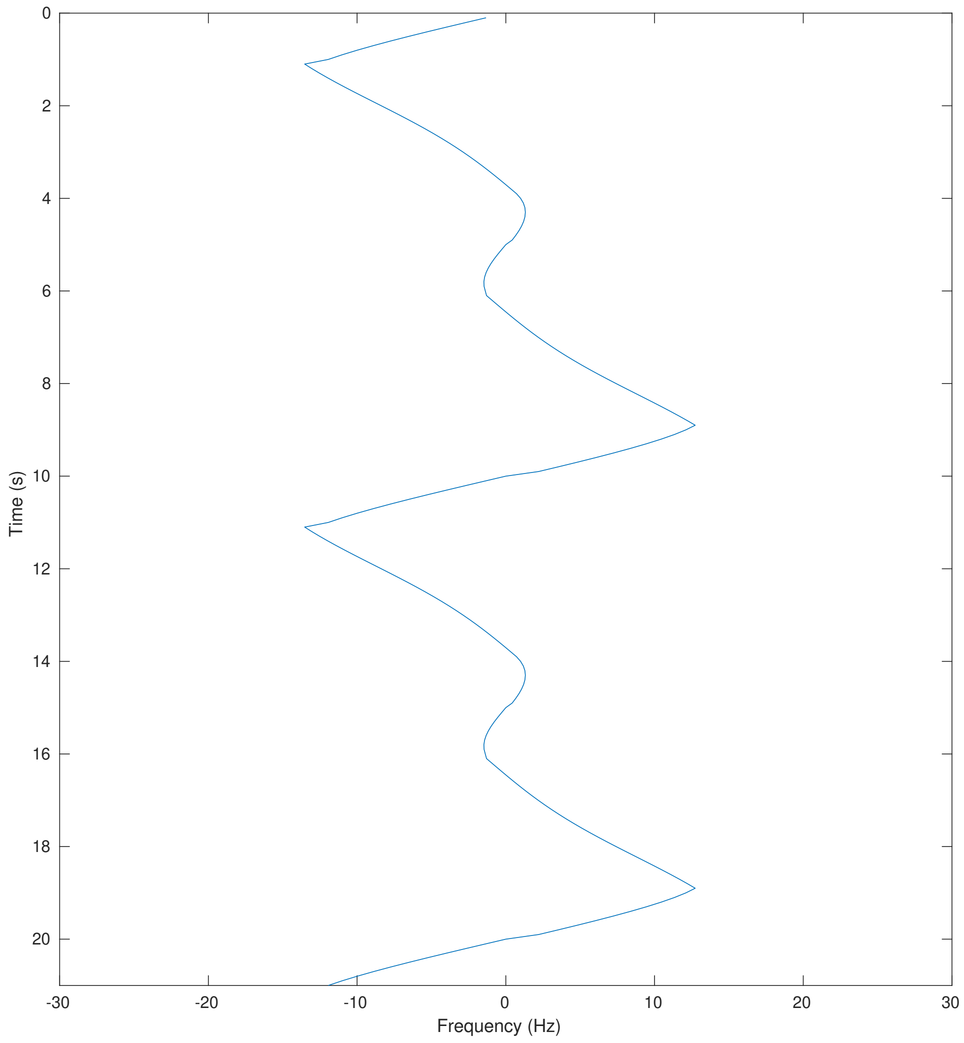


Fig. 5.19: Doppler frequency simulated at the NW antenna.

5.2.7 October 2, 2020

For these tests, the transmitter was placed right by the door to the hall as depicted in Figure 2.6. The receive antennas were set up as mentioned in Section 5.2.6. The position of the transmitter relative to the SW antenna was 10.7 m to the east and 0.9 m to the south. These tests were the first successful hall tests. The setup for these tests is recorded in Tables 5.21 – 5.23 and the tests are recorded in Table 5.24. Prior tests had all failed due to the lack of power on the transmitted signal. An additional amplifier was added to the system to increase the transmit power.

These tests were successful in extracting the Doppler path in a more realistic system.

Table 5.21: Device Parameters, October 2, 2020

Transmitter	B200 mini
Transmitter gain	85 dB
Receiver 1	B210-1, RF A, Tx/Rx
Receiver 1 gain	45 dB
Receiver 1 antenna	SC building antenna
Receiver 2	B210-2, RF A, Tx/Rx
Receiver 2 gain	56 dB
Receiver 2 antenna	SW building antenna
Receiver 3	B200, Tx/Rx
Receiver 3 gain	80 dB
Receiver 3 antenna	NW building antenna
Number of amplifiers	2
Sample rate	62500 Samples/s
Interpolation/Decimation rate	4
Carrier frequency	905 MHz
Master clock rate	n/a

Table 5.22: Symbol Timing Synchronization Parameters, October 2, 2020

Samples per symbol	8
Excess bandwidth	0.5
Matched filter length	161
TED	sgn(y[n])y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.02
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.23: Carrier phase synchronization parameters, October 2, 2020

B_nT_s	90 m
ζ	0.707
K_p	0.5
K_0	1

Table 5.24: Tests, October 2, 2020

Test	Metal	Description
1	Yes	Moving between the SW and SC antennas
2	No	Moving between the SW and SC antennas
3	Yes	Moving between the SW antenna and transmit antenna
4	No	Moving between the SW antenna and transmit antenna

However, as Figure 5.20 and Figure 5.21 show, the distance from the transmitter to a receiver makes a difference. The NW antenna was not able to pick up a strong enough Doppler echo to extract the correct Doppler frequency path.

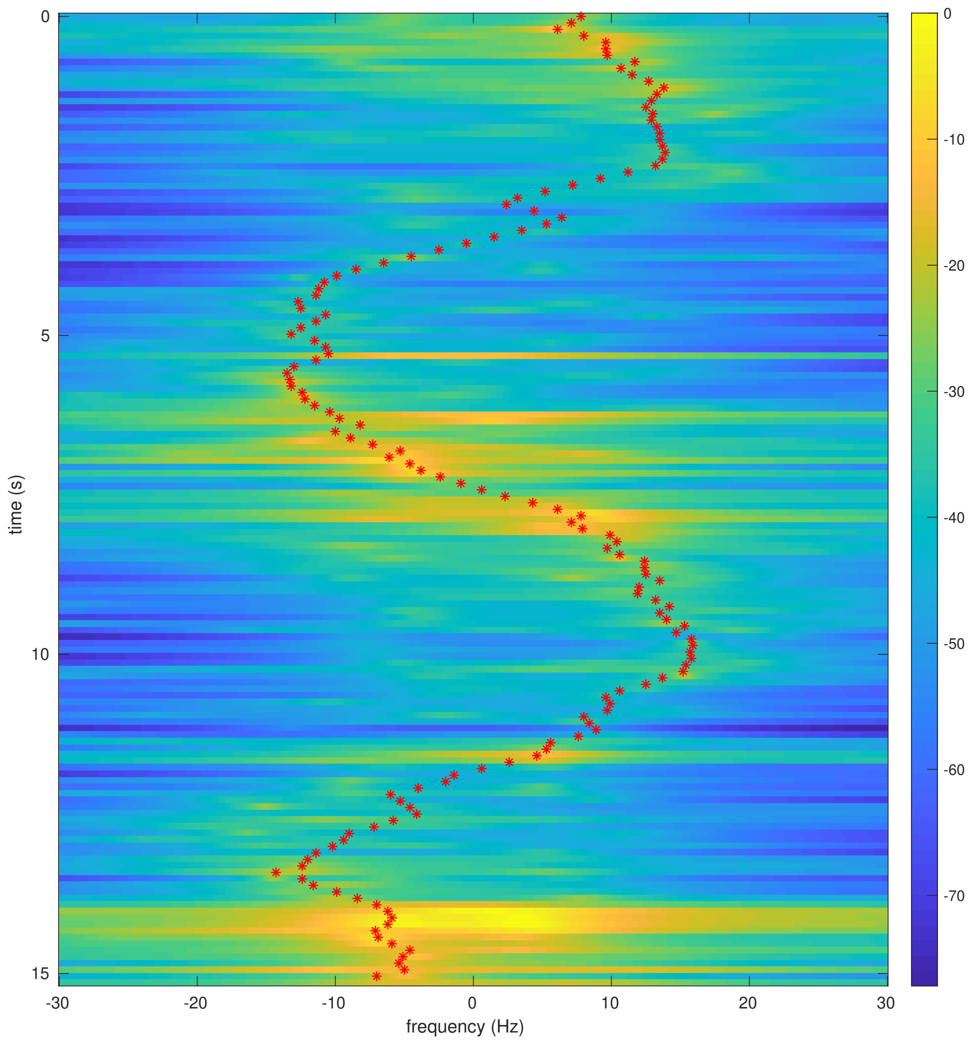


Fig. 5.20: Test 1 with optimal path plotted on MUSIC spectrum, SW antenna.

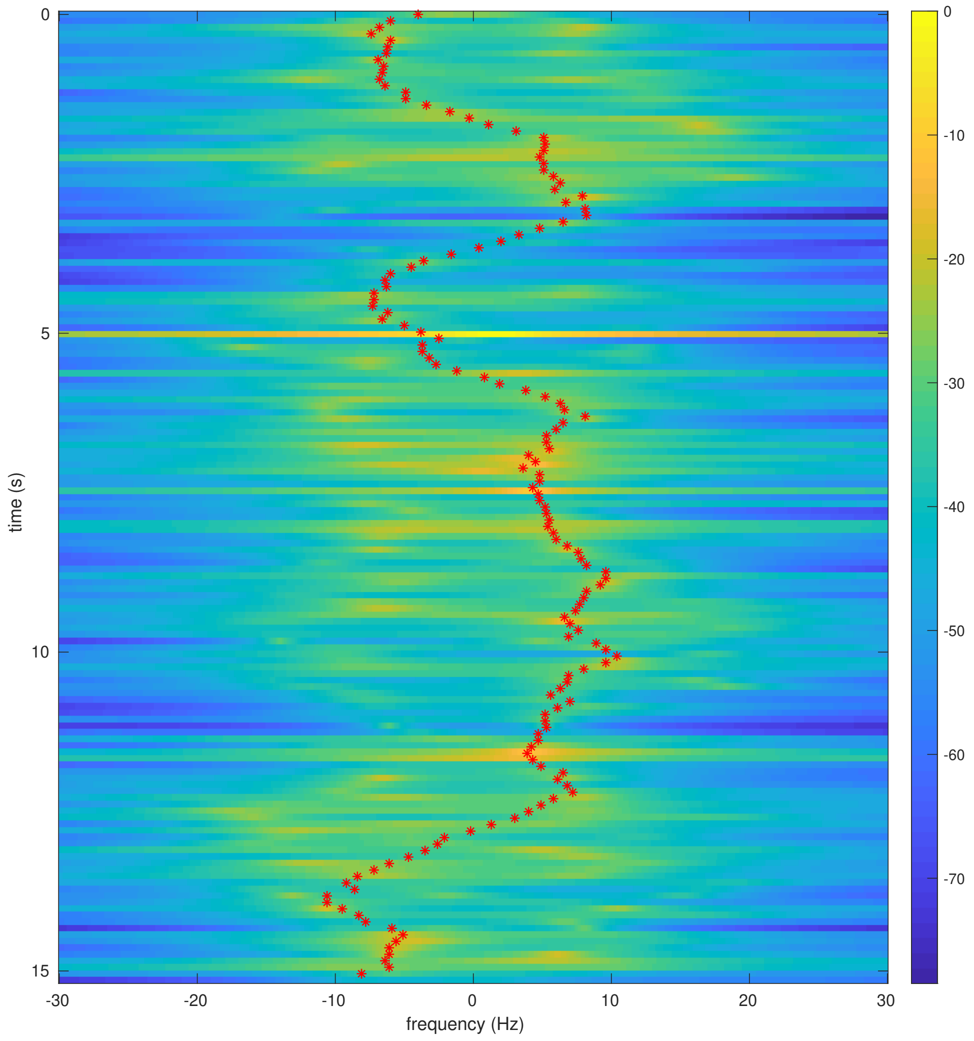


Fig. 5.21: Test 1 with optimal path plotted on MUSIC spectrum, NW antenna.

5.2.8 October 9, 2020

For these tests, the transmitter antenna was placed right by the door as in the previous test set. The receiver antennas were positioned as described in Section 5.2.6. The position of the transmitter relative to the SW antenna was 10.7 m to the east and 0.9 m to the south. The setup for these tests is recorded in Tables 5.25 – 5.27 and the tests are recorded in Table 5.28.

Table 5.25: Device Parameters, October 9, 2020

Transmitter	B210-1, RF A, Tx/Rx
Transmitter gain	85 dB
Receiver 1	B210-1, RF B, Rx2
Receiver 1 gain	43 dB
Receiver 1 antenna	SC building antenna
Receiver 2	B210-2, RF A, R2x
Receiver 2 gain	55 dB
Receiver 2 antenna	SW building antenna
Receiver 3	B200, Rx2
Receiver 3 gain	60 dB
Receiver 3 antenna	NC building antenna
Receiver 4	B200 mini, Rx2
Receiver 4 gain	77 dB
Receiver 4 antenna	NW building antenna
Number of amplifiers	2
Sample rate	62500 Samples/s
Interpolation/Decimation rate	4
Carrier frequency	905 MHz
Master clock rate	n/a

Table 5.26: Symbol Timing Synchronization Parameters, October 9, 2020

Samples per symbol	8
Excess bandwidth	0.5
Matched filter length	161
TED	sgn(y[n])y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.02
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.27: Carrier phase synchronization parameters, October 9, 2020

B_nT_s	90 m
ζ	0.707
K_p	0.5
K_0	1

Table 5.28: Tests, October 9, 2020

Test	Metal	Description
1	Yes	Moving between the SW and SC antennas
2	No	Moving between the SW and SC antennas
3	Yes	Moving between the SW and NW antennas
4	No	Moving between the SW and NW antennas
5	Yes	Moving between the NC and NW antennas
6	No	Moving between the NC and NW antennas
7	Yes	Moving in a clockwise circle in the middle of the three central rooms depicted in figure 2.6
8	No	Moving in a clockwise circle in the middle of the three central rooms depicted in figure 2.6

These tests were the first with the balance factor correction added to the BCJR algorithm. This helped overcome situations where the Doppler was at 0 Hz. Figures 5.22 – 5.25 show the Doppler at all four of the antennas for Test 1. The NC and SC antenna's extracted paths are similar to the simulated paths. However, the NW and SW paths do not compare as well.

All of these paths were generated using the same parameters. It is usually more appropriate to use different parameters for each antenna when different devices are used at each antenna. This is primarily due to each device having a different carrier frequency offset. It would be more ideal to use a single device or multiple of the same device tied to the same clock.

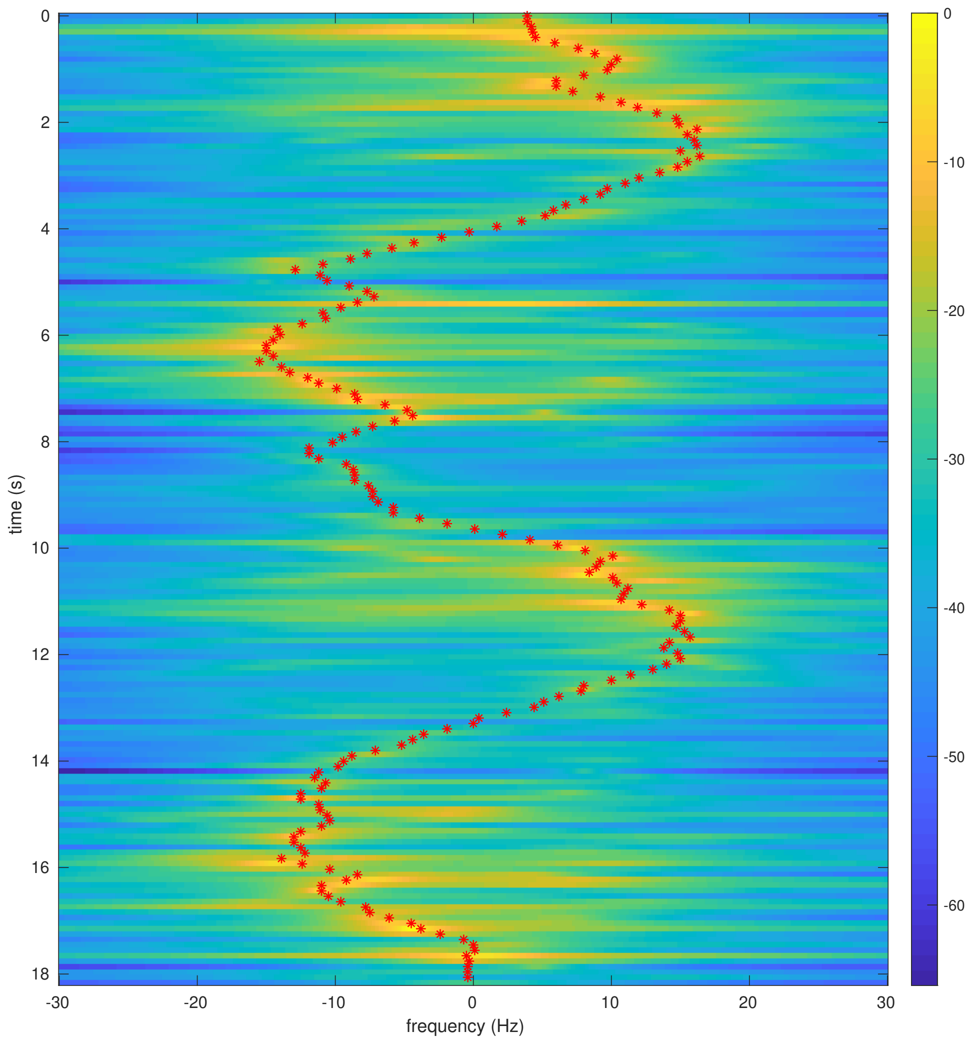


Fig. 5.22: Test 1 with optimal path plotted on MUSIC spectrum, SC antenna.

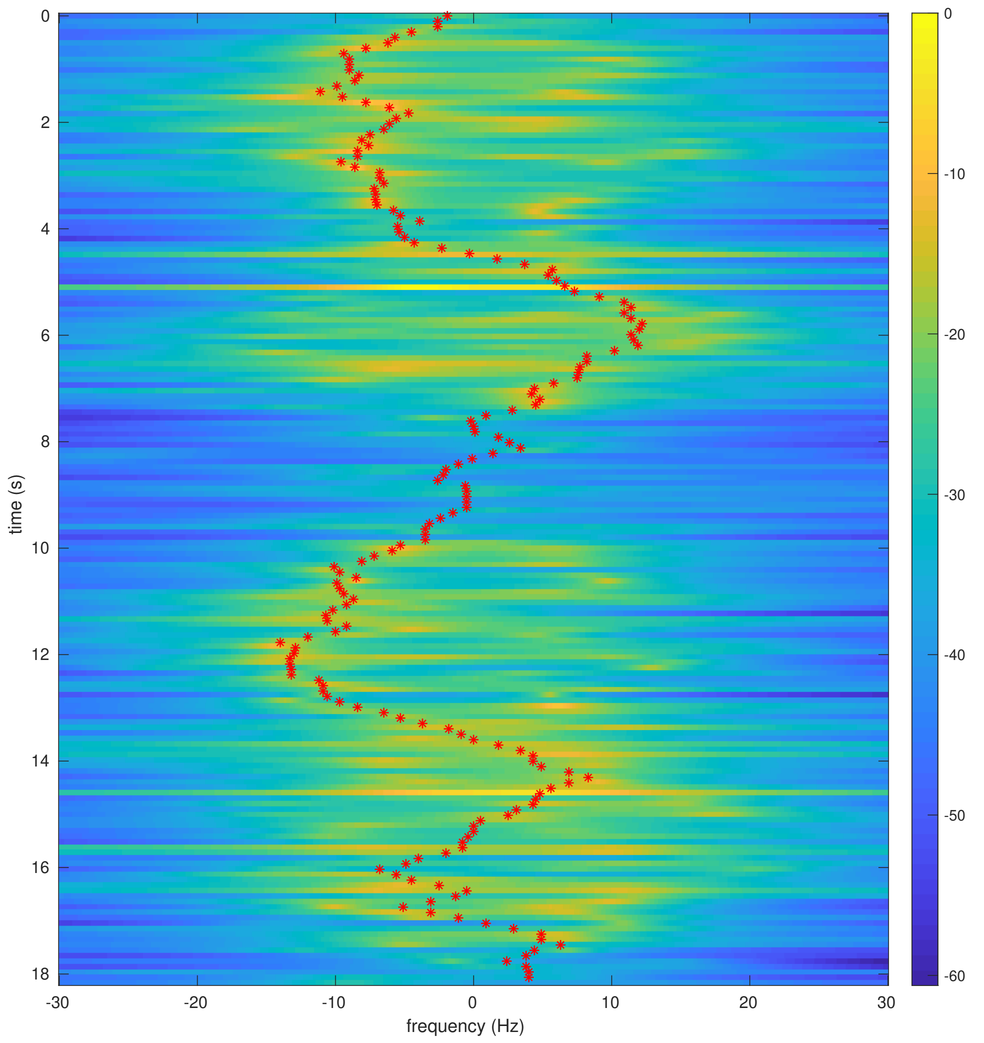


Fig. 5.23: Test 1 with optimal path plotted on MUSIC spectrum, SW antenna.

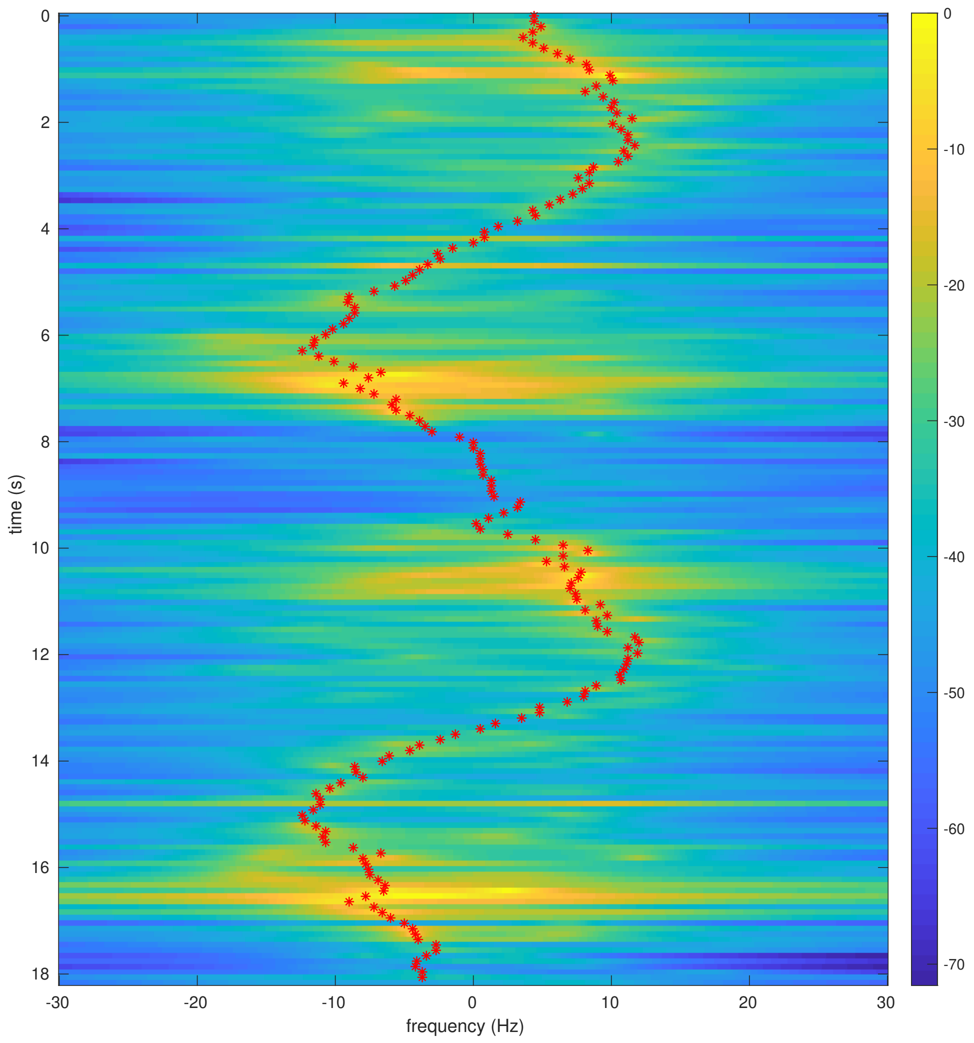


Fig. 5.24: Test 1 with optimal path plotted on MUSIC spectrum, NC antenna.

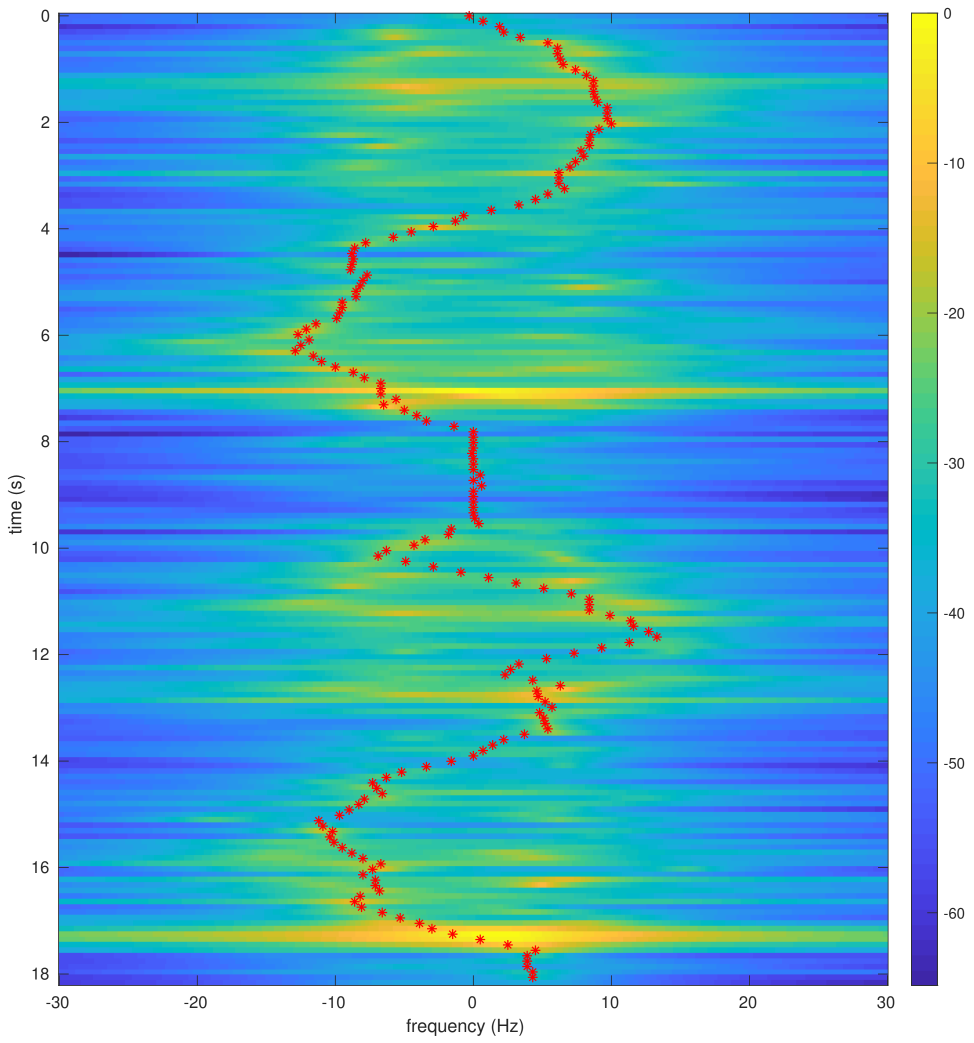


Fig. 5.25: Test 1 with optimal path plotted on MUSIC spectrum, NW antenna.

5.2.9 November 6, 2020

These tests moved the transmitter antenna further away from the door. The receiver antennas were positioned as described in Section 5.2.6. The transmitter was 10.4 m east and 5.8 m south of the SW antenna. The setup for these tests is recorded in Tables 5.29 – 5.31 and the tests are recorded in Table 5.32.

Figures 5.26 – 5.29 are the results of the Doppler extraction for Test 1. These results are encouraging, as both the SW and SC antenna paths are regular back and forth paths.

Table 5.29: Device Parameters, November 6, 2020

Transmitter	B210-1, RF A, Tx/Rx
Transmitter gain	85 dB
Receiver 1	B210-1, RF B, Rx2
Receiver 1 gain	51 dB
Receiver 1 antenna	SC building antenna
Receiver 2	B210-2, RF A, R2x
Receiver 2 gain	76 dB
Receiver 2 antenna	NW building antenna
Receiver 3	B200, Rx2
Receiver 3 gain	57 dB
Receiver 3 antenna	NC building antenna
Receiver 4	B200 mini, Rx2
Receiver 4 gain	75 dB
Receiver 4 antenna	SW building antenna
Number of amplifiers	2
Sample rate	62500 Samples/s
Interpolation/Decimation rate	4
Carrier frequency	905 MHz
Master clock rate	n/a

Table 5.30: Symbol Timing Synchronization Parameters, November 6, 2020

Samples per symbol	8
Excess bandwidth	0.5
Matched filter length	161
TED	sgn(y[n])y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.02
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.31: Carrier phase synchronization parameters, November 6, 2020

B_nT_s	90 m
ζ	0.707
K_p	0.5
K_0	1

Table 5.32: Tests, November 6, 2020

Test	Metal	Description
1	Yes	Moving between the SW and SC antennas
2	No	Moving between the SW and SC antennas
3	Yes	Moving between the SW and NW antennas
4	No	Moving between the SW and NW antennas
5	Yes	Moving between the NC and NW antennas
6	No	Moving between the NC and NW antennas

However, the NW and NC paths are very poor. This is unsurprising as moving the transmitter further away would cause the Doppler echo to be even weaker when it reaches these antennas. The balance factor pulls the Doppler path to 0 Hz at the NC antenna because the path is too weak. This suggests the NC antenna is outside of the maximum range the system can operate in.

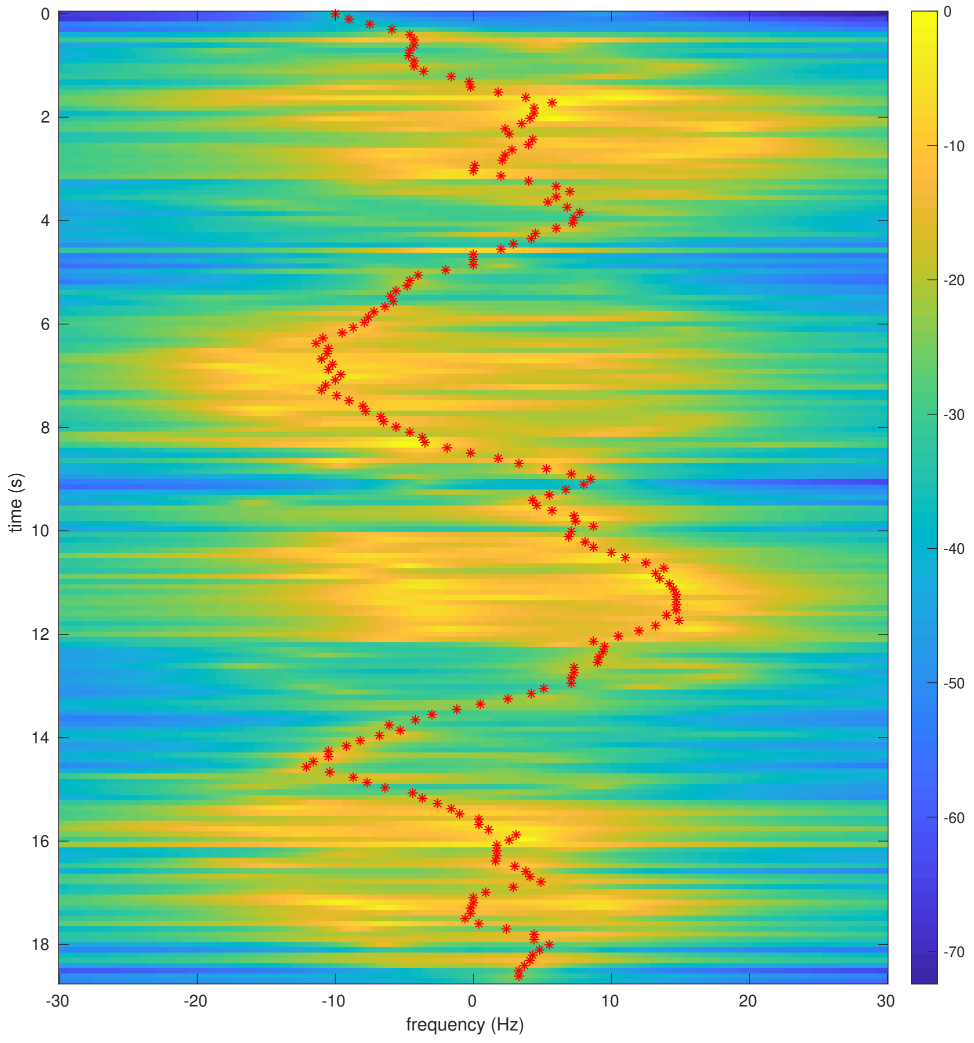


Fig. 5.26: Test 1 with optimal path plotted on MUSIC spectrum, SC antenna.

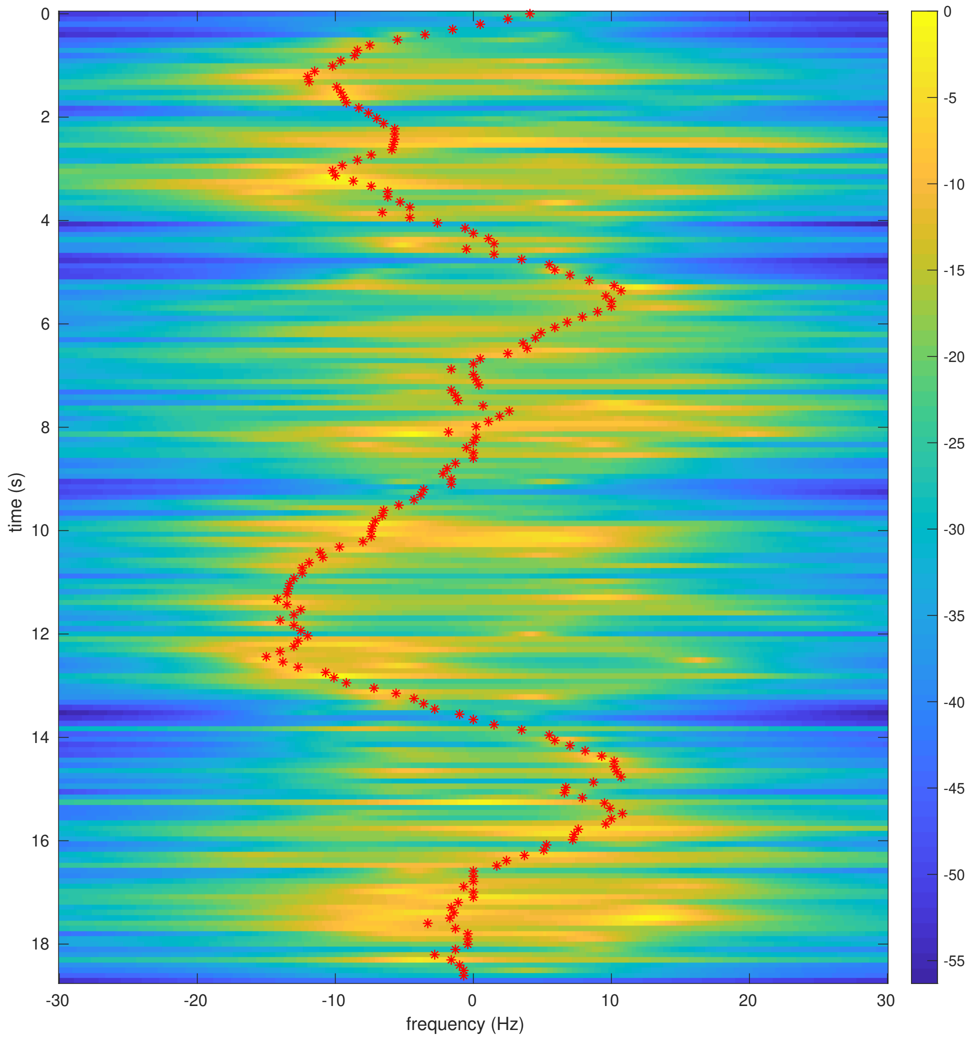


Fig. 5.27: Test 1 with optimal path plotted on MUSIC spectrum, SW antenna.

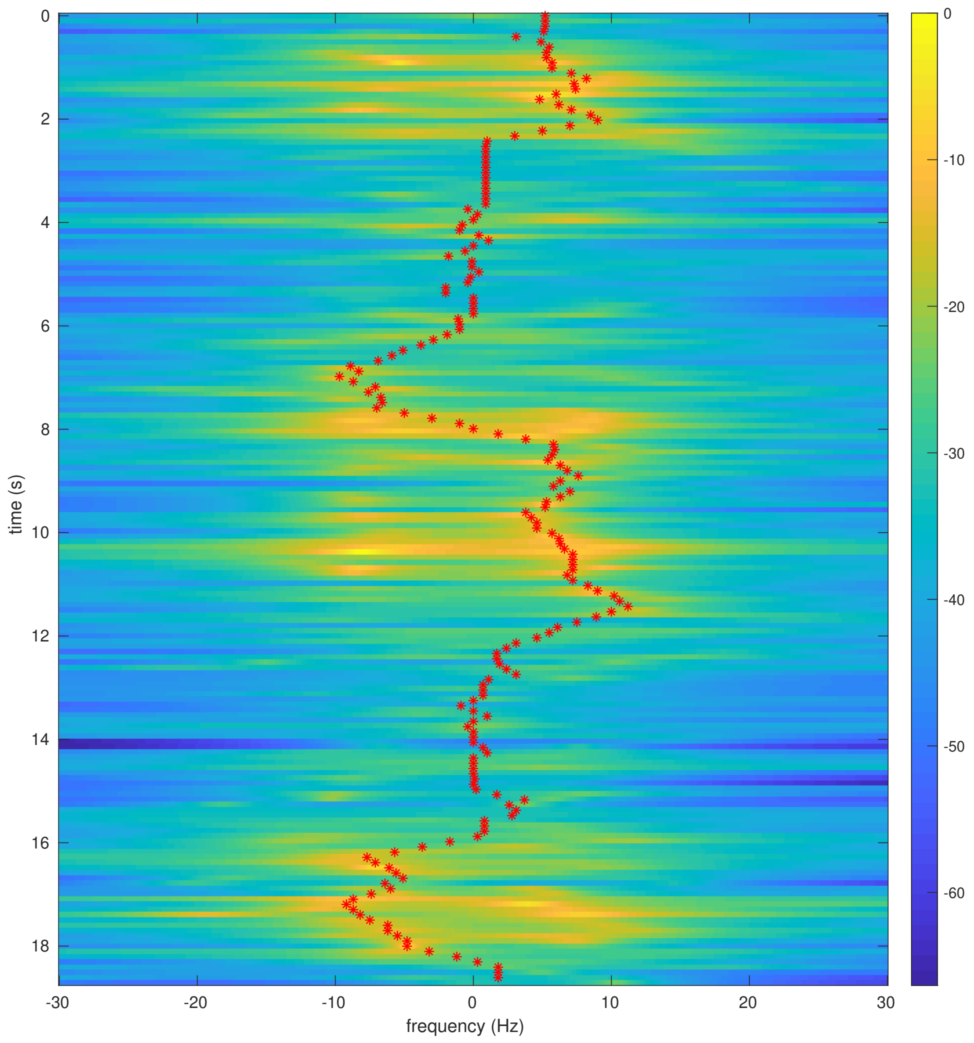


Fig. 5.28: Test 1 with optimal path plotted on MUSIC spectrum, NC antenna.

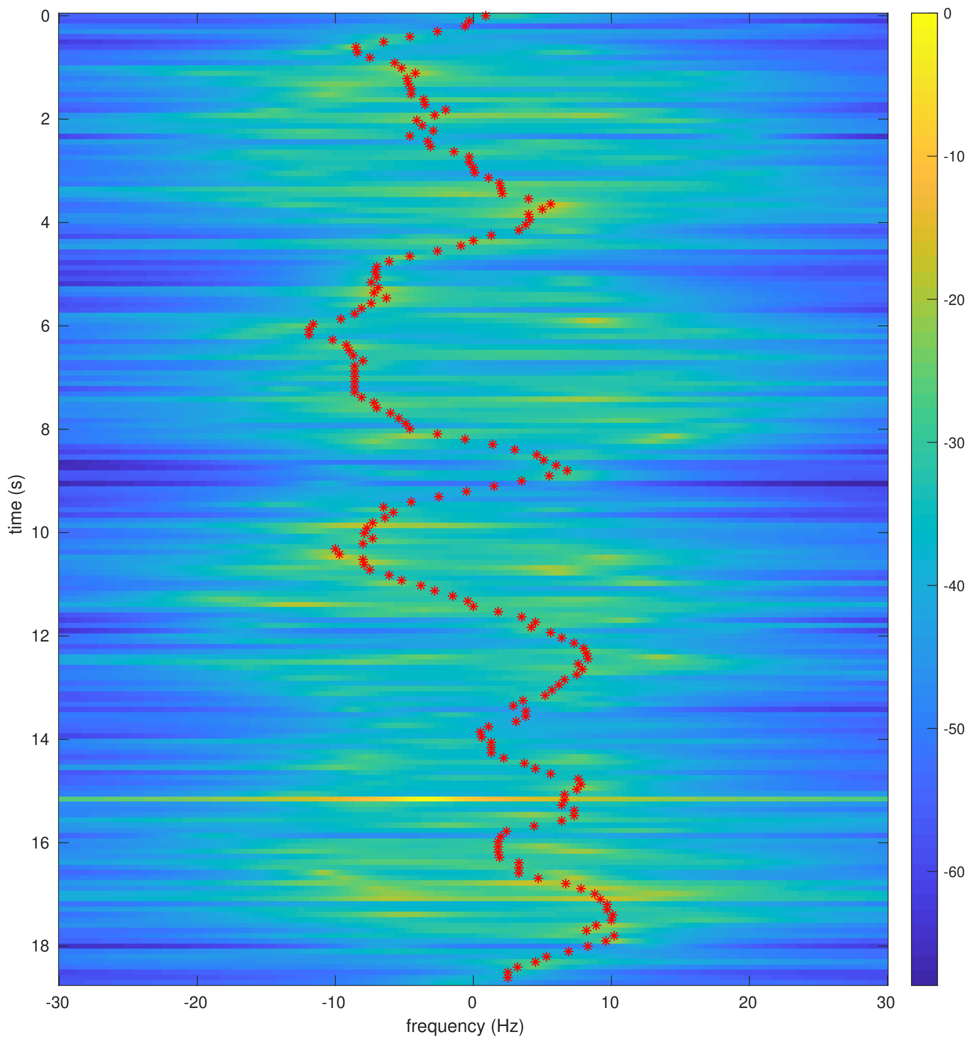


Fig. 5.29: Test 1 with optimal path plotted on MUSIC spectrum, NW antenna.

5.2.10 November 13, 2020

These tests moved the transmitter antenna to a more centralized location. The transmitter was 7.01 m east and 6.4 m north of the SW antenna. This position is in the middle room of the three central rooms of Figure 2.6. The receiver antennas were positioned as described in Section 5.2.6. The setup for these tests is recorded in Tables 5.33 – 5.35 and the tests are recorded in Table 5.36.

There were only two tests in this test set, one with metal and one without. Figures

Table 5.33: Device Parameters, November 13, 2020

Transmitter	B00, Tx/Rx
Transmitter gain	85 dB
Receiver 1	B210-1, RF A, Rx2
Receiver 1 gain	48 dB
Receiver 1 antenna	SC building antenna
Receiver 2	B210-2, RF A, R2x
Receiver 2 gain	64 dB
Receiver 2 antenna	NW building antenna
Receiver 3	B200 mini-1, Rx2
Receiver 3 gain	63 dB
Receiver 3 antenna	SW building antenna
Receiver 4	B200 mini-2, Rx2
Receiver 4 gain	85 dB
Receiver 4 antenna	NC building antenna
Number of amplifiers	2
Sample rate	62500 Samples/s
Interpolation/Decimation rate	4
Carrier frequency	905 MHz
Master clock rate	n/a

Table 5.34: Symbol Timing Synchronization Parameters, November 13, 2020

Samples per symbol	8
Excess bandwidth	0.5
Matched filter length	161
TED	sgn(y[n])y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.02
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.35: Carrier phase synchronization parameters, November 13, 2020

B_nT_s	90 m
ζ	0.707
K_p	0.5
K_0	1

Table 5.36: Tests, November 13, 2020

Test	Metal	Description
1	Yes	Moving between the SW and SC antennas
2	No	Moving between the SW and SC antennas

5.30 – 5.37 compare Test 1 to Test 2 at each antenna. For the SC, SW, and NC antennas, the cases with and without metal are very similar to each other. Some data appears to be lost in the test without metal, Test 2, but overall the general path seems to follow the assumed Doppler path.

However, the NW antenna appears to receive the weakest signal; the algorithm seems to follow an incorrect path at this antenna. In the test with metal, this incorrect path appears to hover around 0 Hz as the balance factor pulls the Doppler path towards this state. In the case without metal, the balance factor does not seem to be as effective. The Doppler path instead follows what may be a false path. However, the correct path is hard to determine as the transmitter location causes the paths to differ significantly from the simulated paths of Figures 5.16 – 5.19.

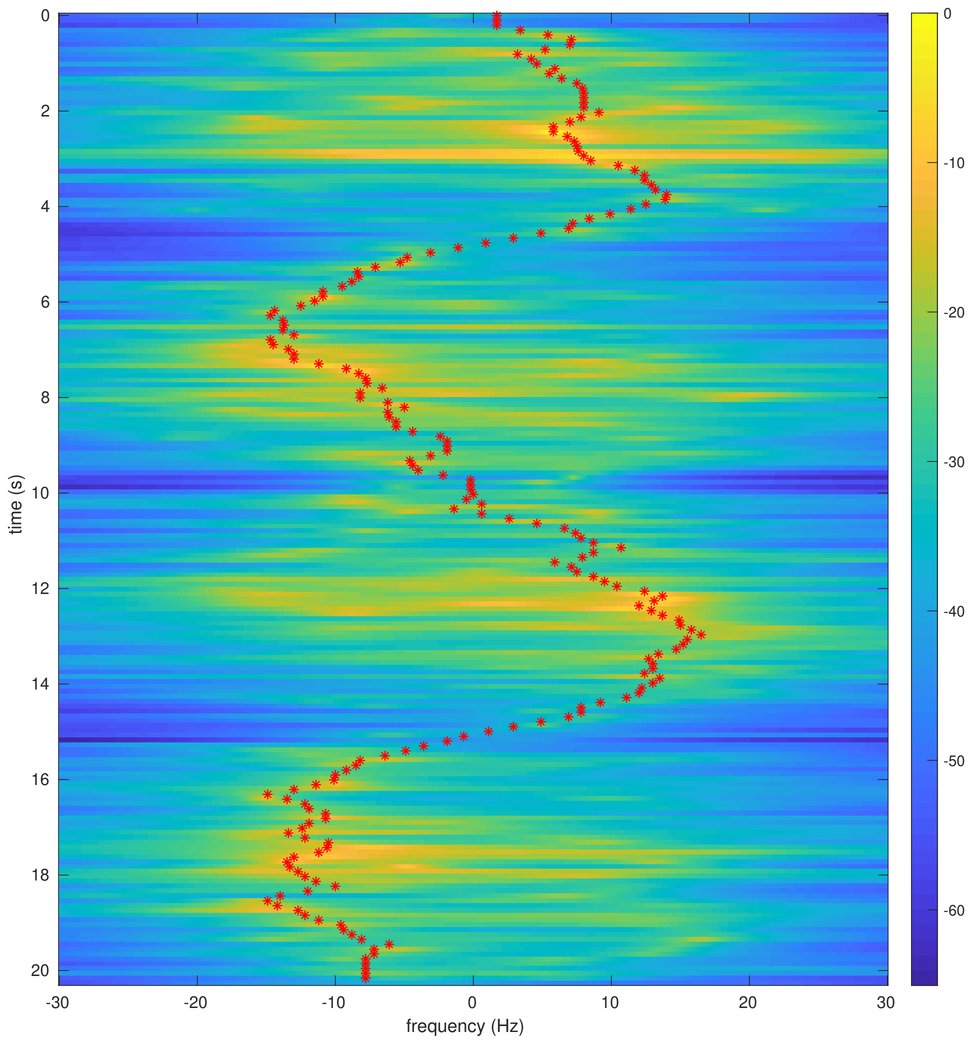


Fig. 5.30: Test 1 with the optimal path plotted on a MUSIC spectrum, SC antenna.

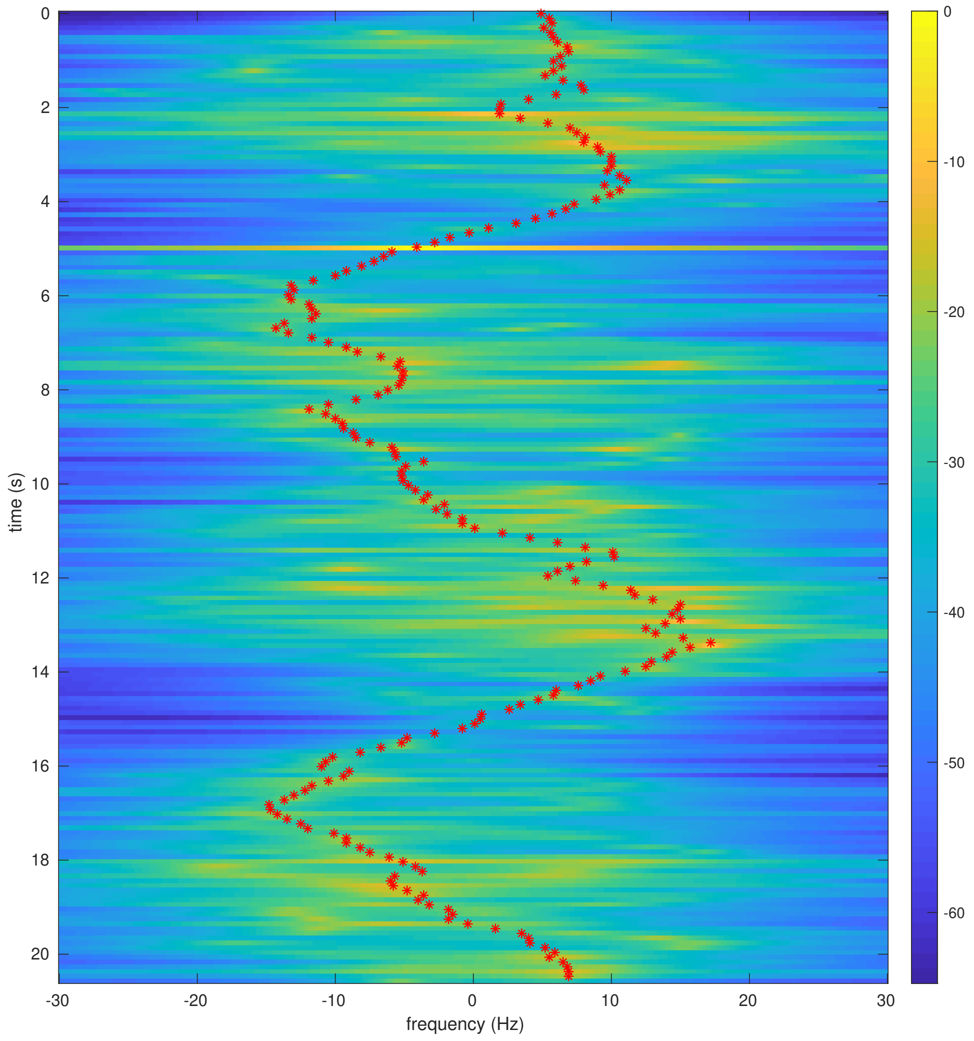


Fig. 5.31: Test 2 with the optimal path plotted on a MUSIC spectrum, SC antenna.

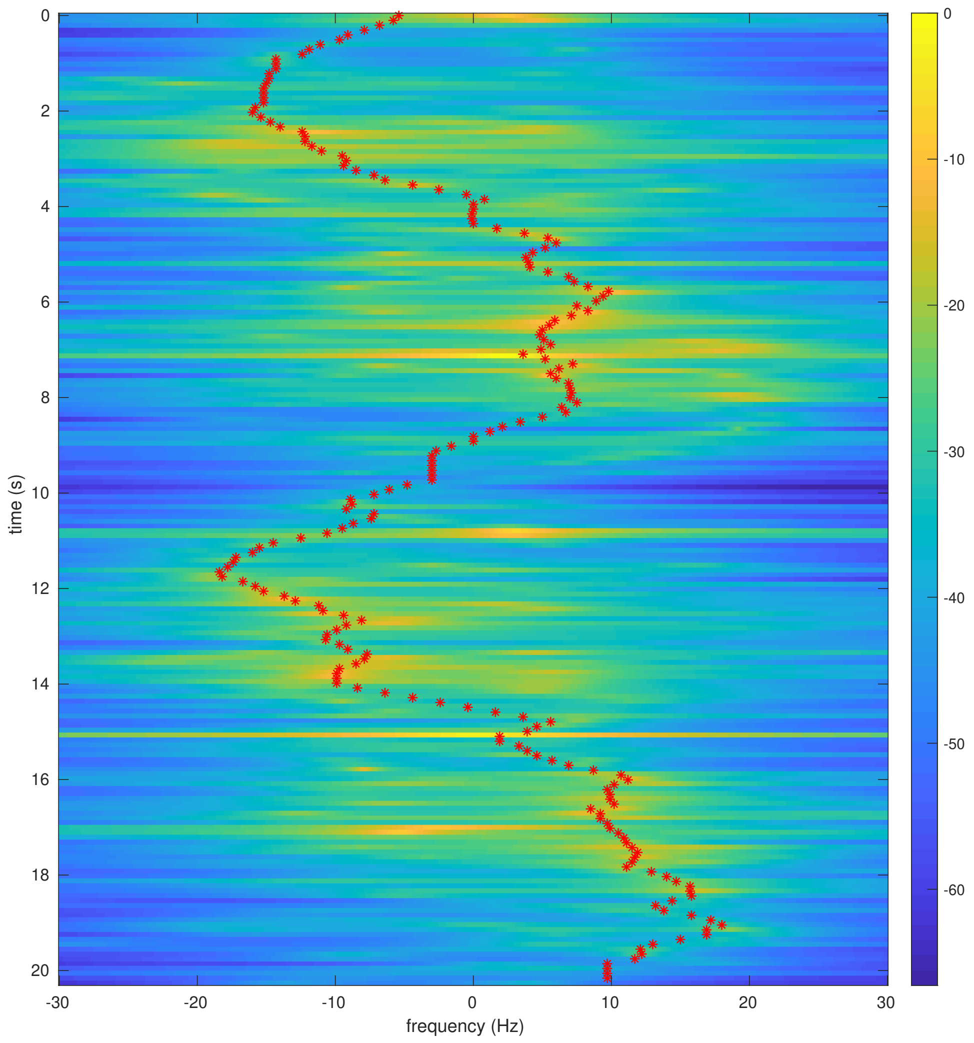


Fig. 5.32: Test 1 with the optimal path plotted on a MUSIC spectrum, SW antenna.

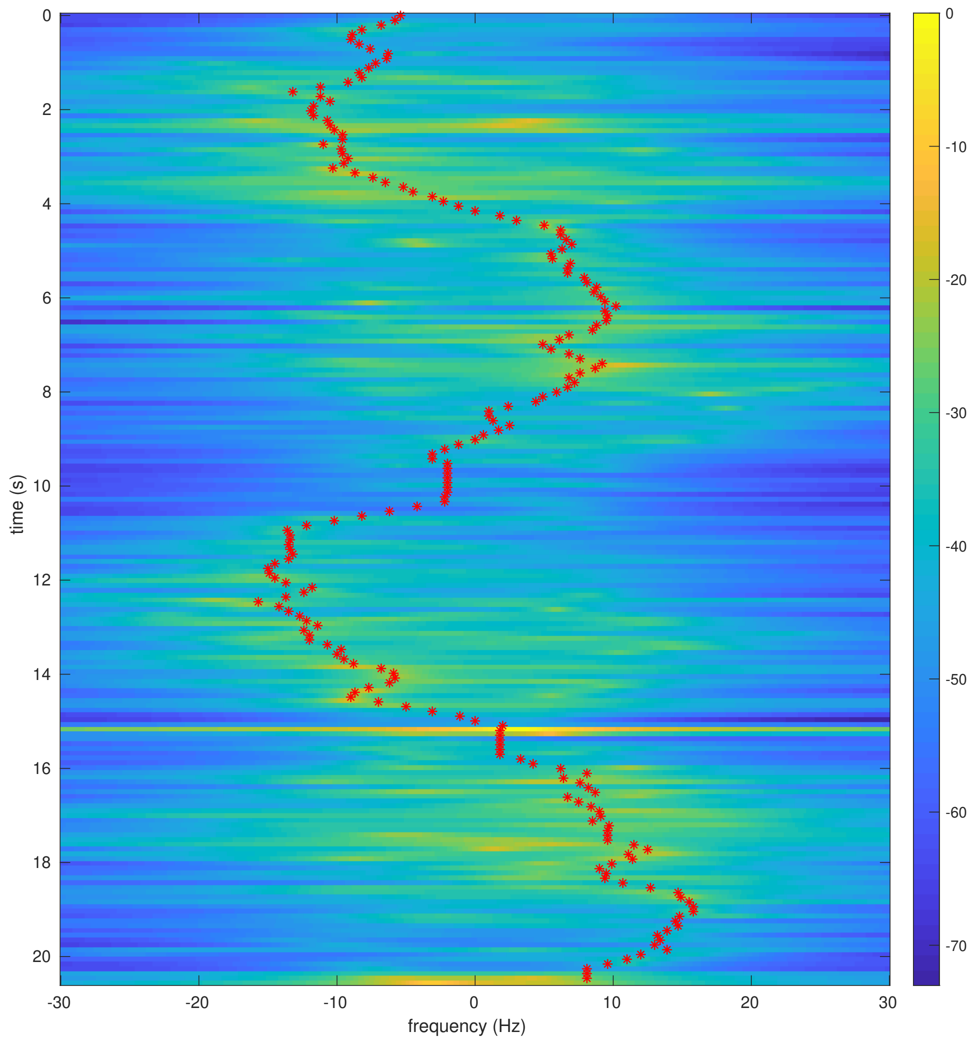


Fig. 5.33: Test 2 with the optimal path plotted on a MUSIC spectrum, SW antenna.

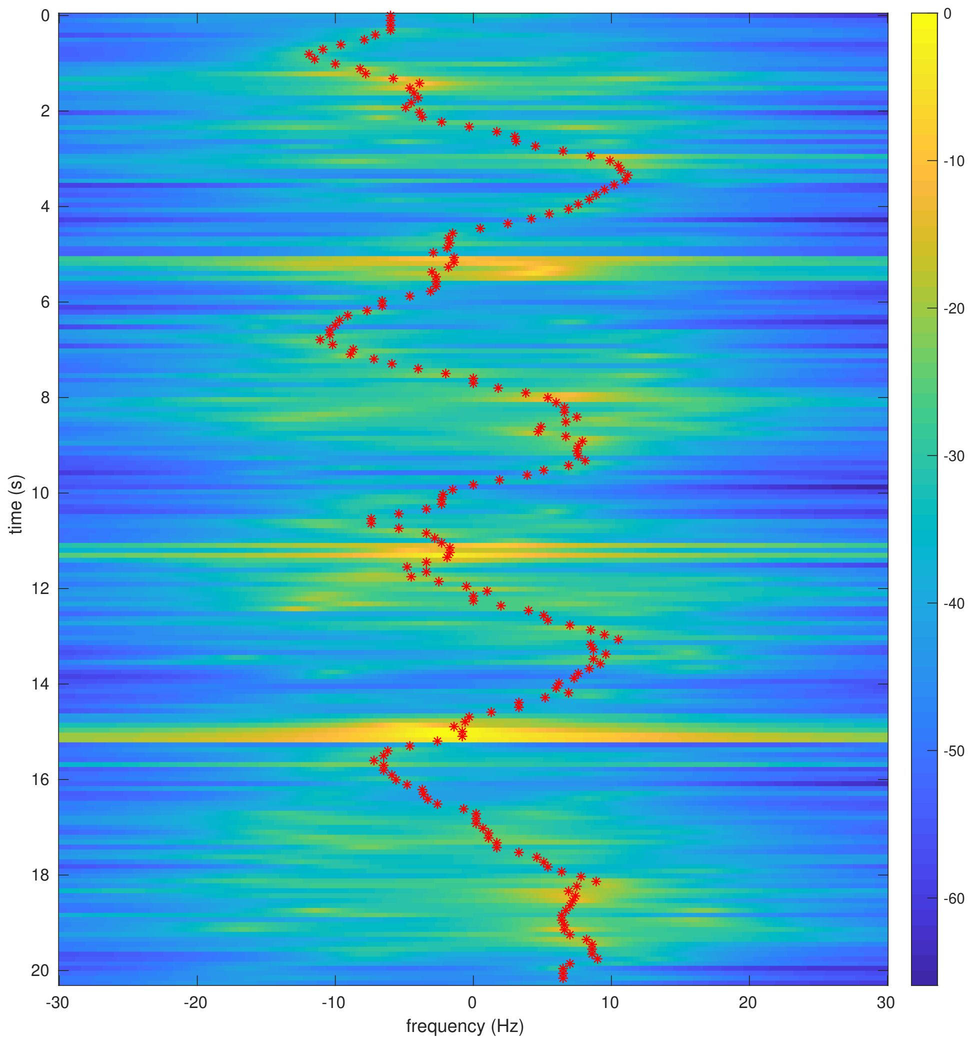


Fig. 5.34: Test 1 with the optimal path plotted on a MUSIC spectrum, NC antenna.

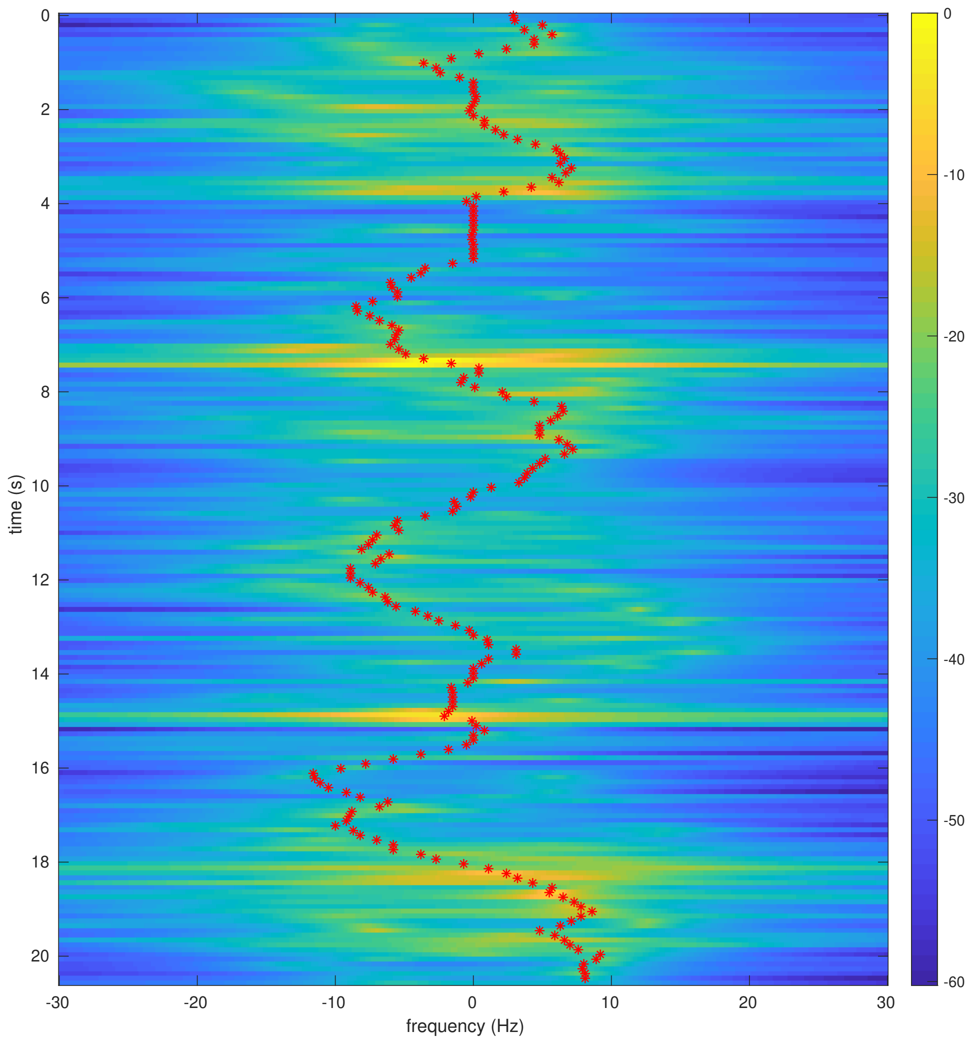


Fig. 5.35: Test 2 with the optimal path plotted on a MUSIC spectrum, NC antenna.

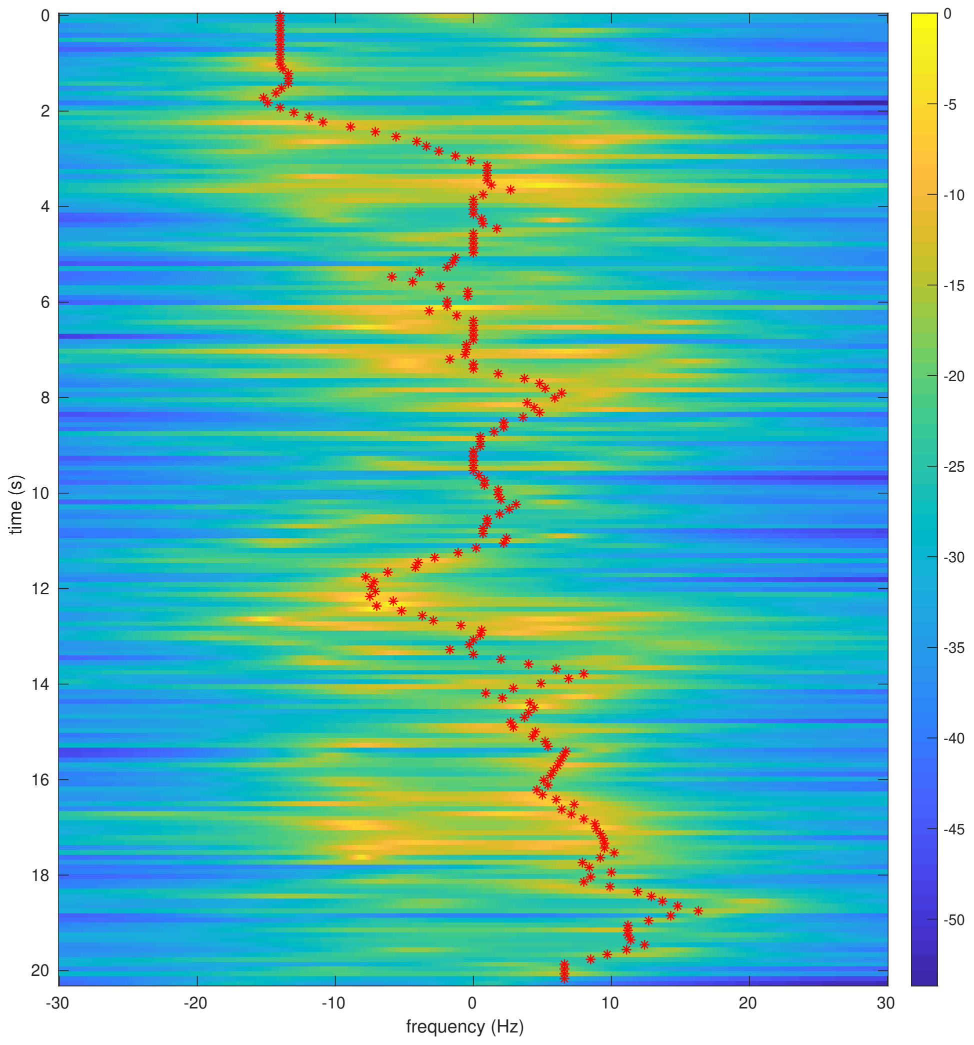


Fig. 5.36: Test 1 with the optimal path plotted on a MUSIC spectrum, NW antenna.

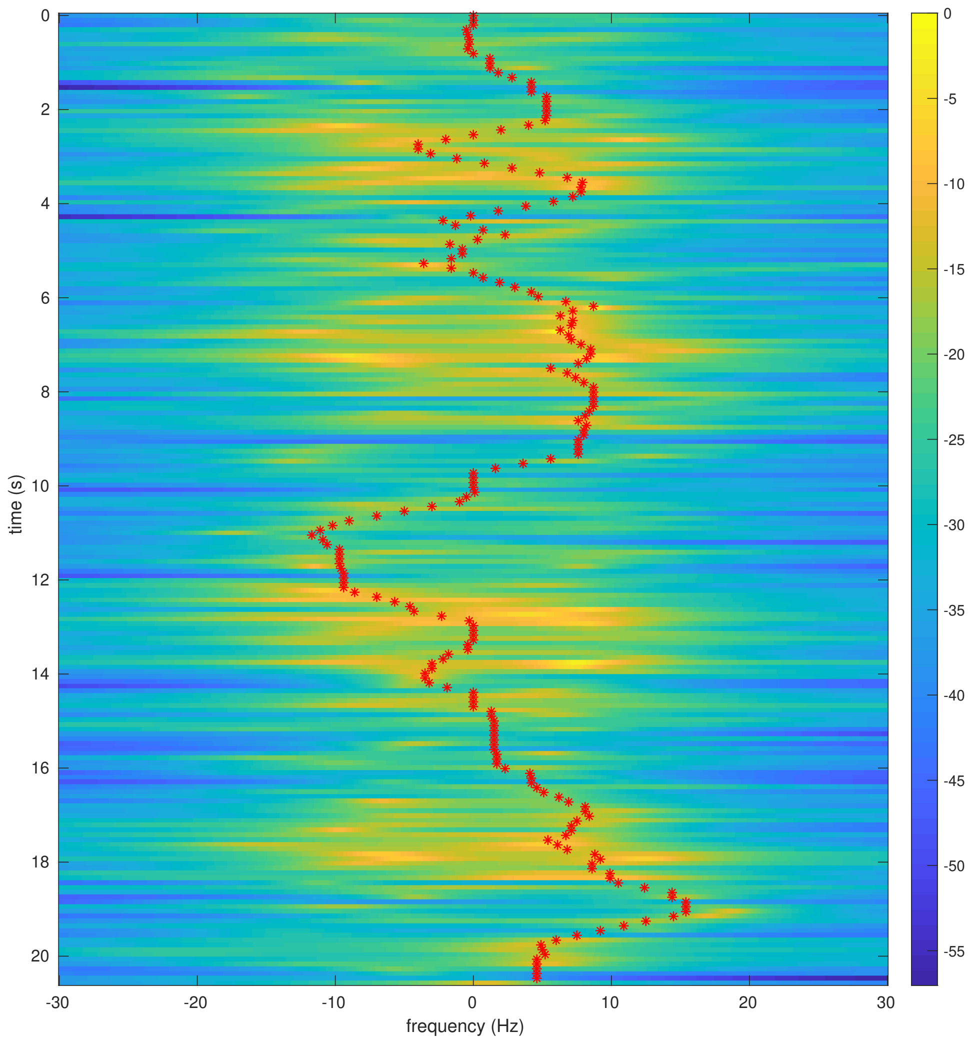


Fig. 5.37: Test 2 with the optimal path plotted on a MUSIC spectrum, NW antenna.

5.2.11 February 12, 2021

The major problem with all the previous tests to this test set was the lack of devices with good clocks. The E320 devices had phase jitter problems on their clocks, and all of the other devices' clocks were also noisy. In addition, each receiver had its own separate device, which had a separate clock from all the other receivers.

The N310s were a solution to this problem. These devices had four channels, each capable of transmitting or receiving data. This allowed one of the devices to act as the receiver for all four antennas simultaneously. In addition, these devices had more consistent clocks.

This test set used one N310 as a transmitter with the transmit antenna placed 8.23 m east and 1.83 m south of the SW antenna. Another N310 was used as the receiver for the NW, SW, NC, and SC antennas.

Figures 5.38 – 5.45 compare the results of Test 0 to the results of Test 4 at each antenna. These results show the N310 devices are more ideal for collecting data.

Table 5.37: Device Parameters, February 12, 2021

Transmitter	N310-2, RF 0, Tx/Rx
Transmitter gain	65 dB
Receiver 1	N310-1, RF 0, Tx/Rx
Receiver 1 gain	65 dB
Receiver 1 antenna	SC building antenna
Receiver 2	N310-1, RF 1, Tx/Rx
Receiver 2 gain	65 dB
Receiver 2 antenna	SW building antenna
Receiver 3	N310-1, RF 2, Tx/Rx
Receiver 3 gain	65 dB
Receiver 3 antenna	NW building antenna
Receiver 4	N310-1, RF 3, Tx/Rx
Receiver 4 gain	65 dB
Receiver 4 antenna	NC building antenna
Number of amplifiers	1
Sample rate	153600 Samples/s
Interpolation/Decimation rate	8
Carrier frequency	905 MHz
Master clock rate	122.88 MHz

Table 5.38: Symbol Timing Synchronization Parameters, February 12, 2021

Samples per symbol	8
Excess bandwidth	0.35
Matched filter length	161
TED	sgn(y[n])y'[n] Maximum Likelihood
Expected TED gain	1
Loop bandwidth	0.02
Damping factor	1.3
Maximum deviation	1.5
Interpolating resampler	MMSE, 8-tap FIR

Table 5.39: Carrier phase synchronization parameters, February 12, 2021

B_nT_s	90 m
ζ	0.707
K_p	0.5
K_0	1

The similarities between the two tests at each antenna, Test 0 with metal and Test 4 without metal, suggest each path is approaching the correct path in both cases. Comparing them to the optimal paths of the simulated paths in Figures 5.16 – 5.19 also show a more similar resemblance than the previous tests had.

Table 5.40: Tests, February 12, 2021

Test	Metal	Description
0	Yes	Moving between the SW and SC antennas, running
1	Yes	Moving between the SW and SC antennas, walking
2	Yes	Moving between the SW and NW antennas, running
3	Yes	Moving between the SW and NW antennas, walking
4	No	Moving between the SW and SC antennas, running
5	No	Moving between the SW and SC antennas, walking
6	No	Moving between the SW and NW antennas, running
7	No	Moving between the SW and NW antennas, walking

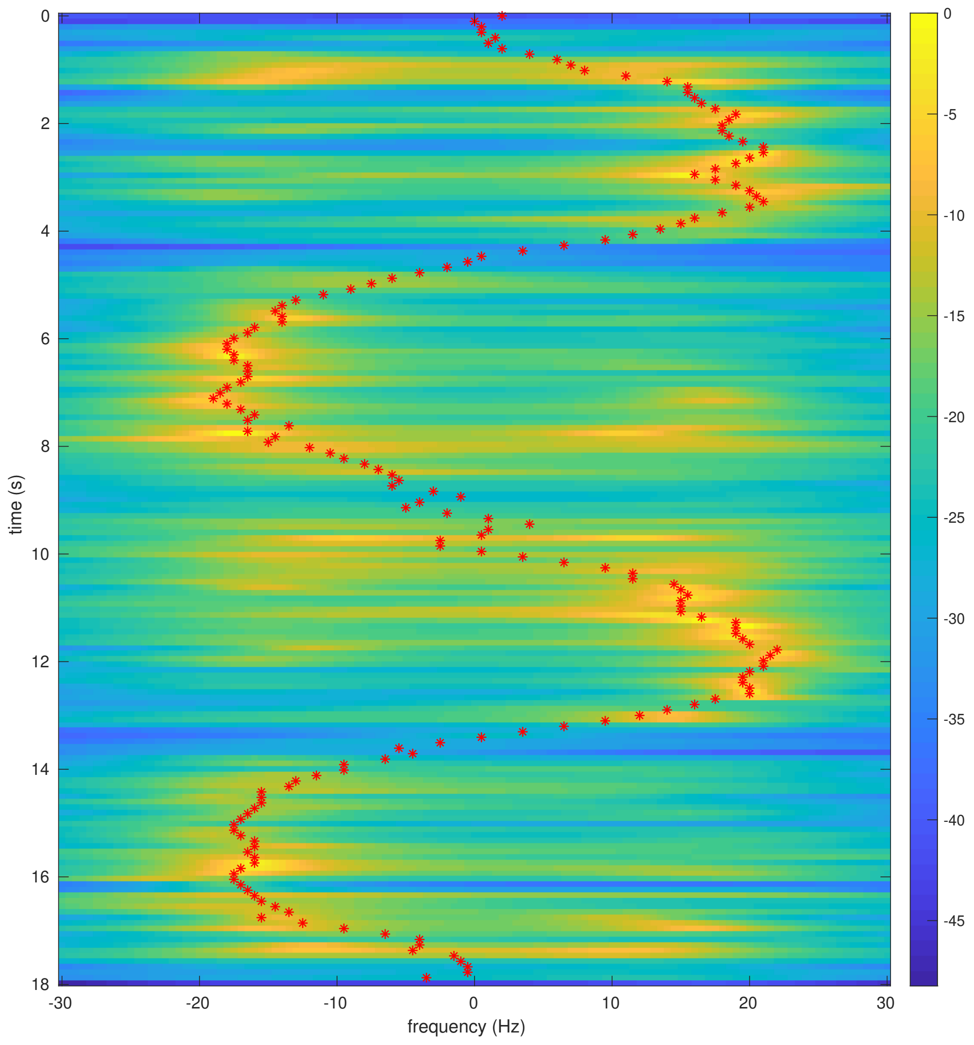


Fig. 5.38: Test 0 with the optimal path plotted on a MUSIC spectrum, SC antenna.

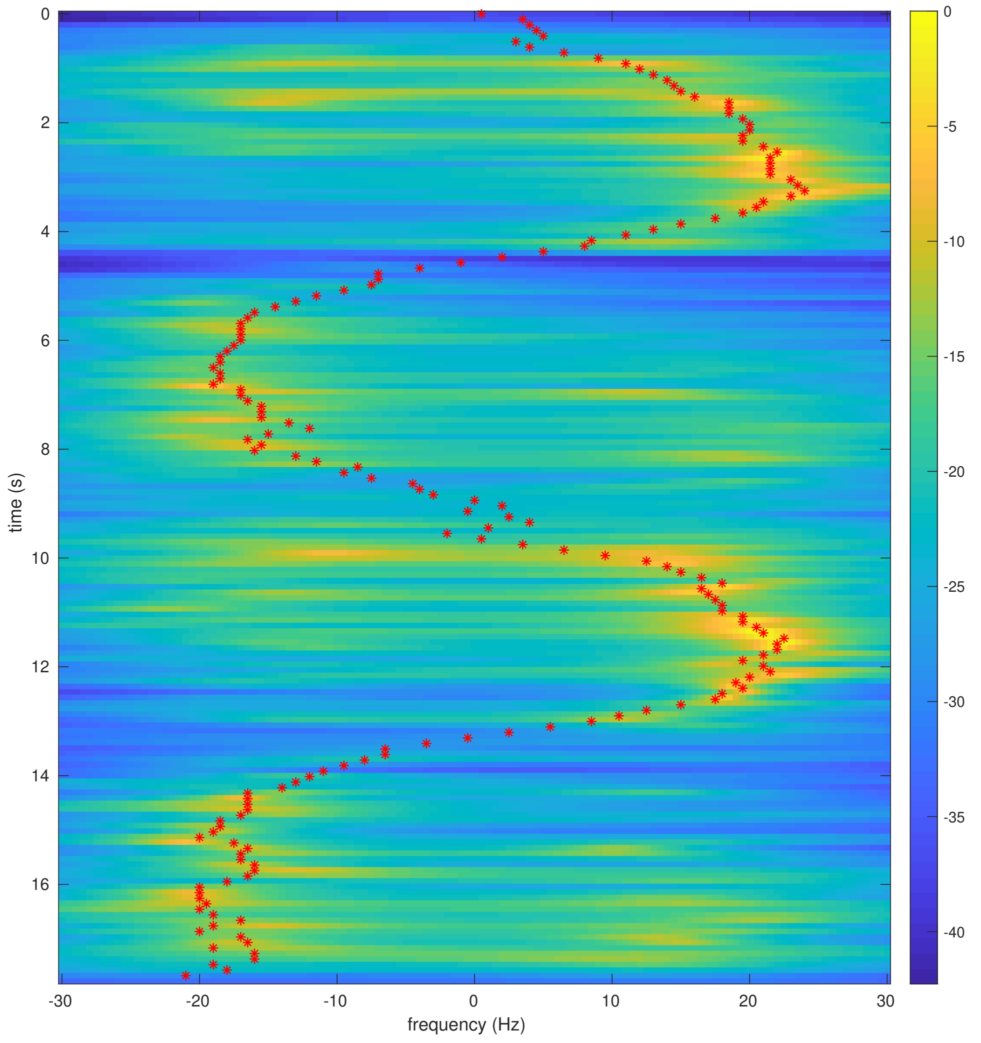


Fig. 5.39: Test 4 with the optimal path plotted on a MUSIC spectrum, SC antenna.

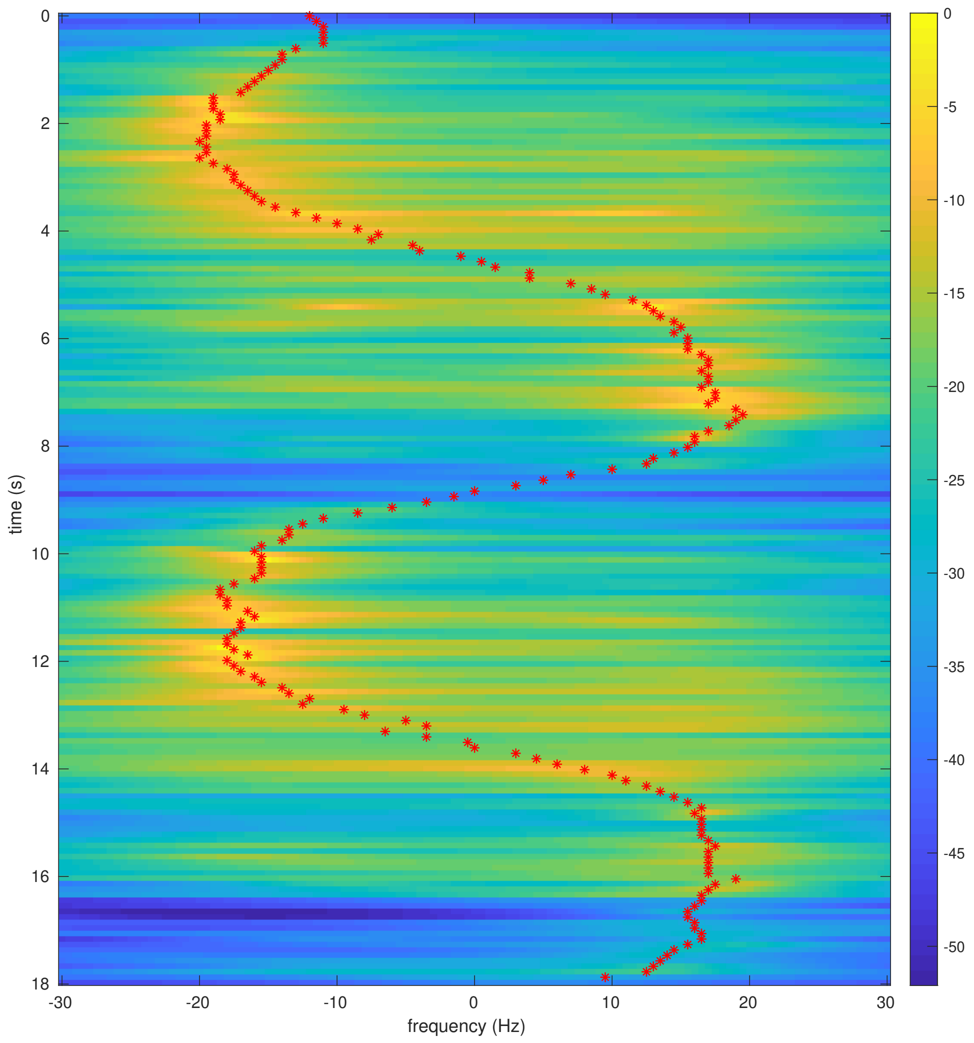


Fig. 5.40: Test 0 with the optimal path plotted on a MUSIC spectrum, SW antenna.

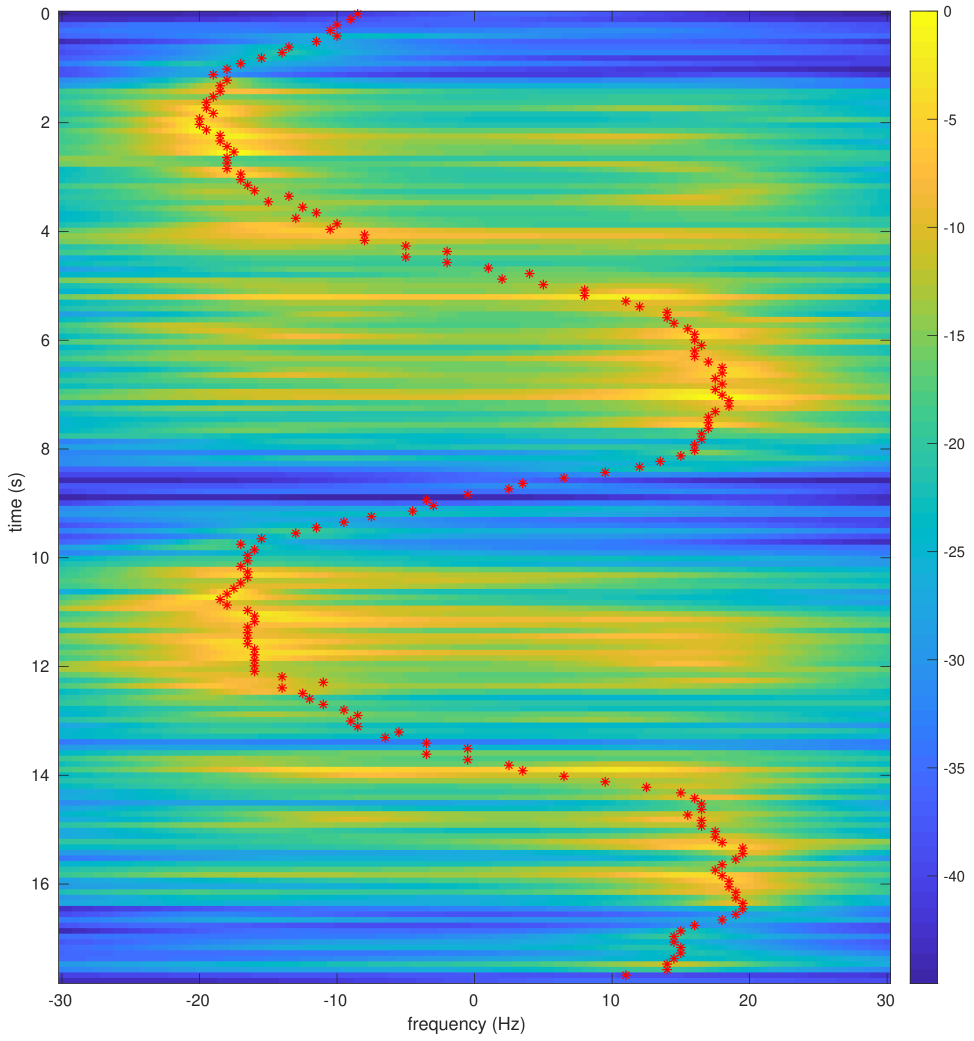


Fig. 5.41: Test 4 with the optimal path plotted on a MUSIC spectrum, SW antenna.

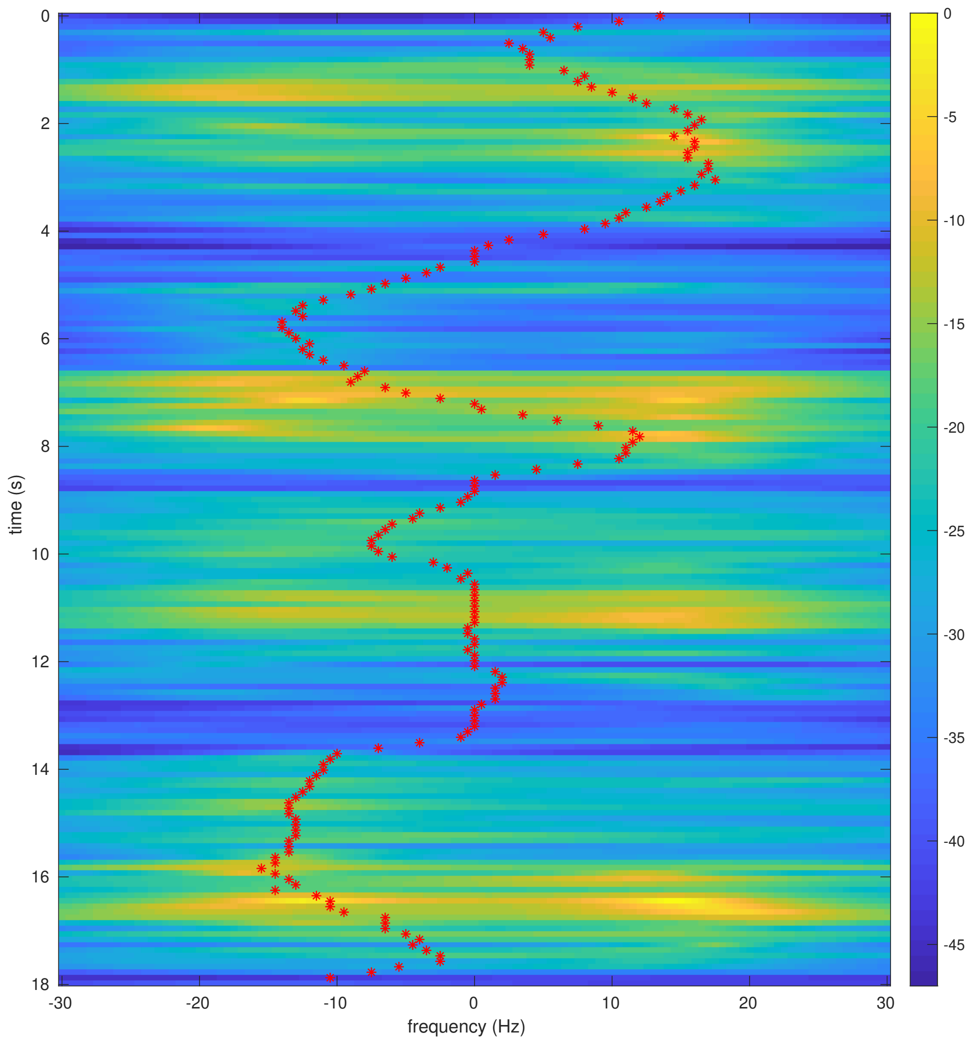


Fig. 5.42: Test 0 with the optimal path plotted on a MUSIC spectrum, NC antenna.

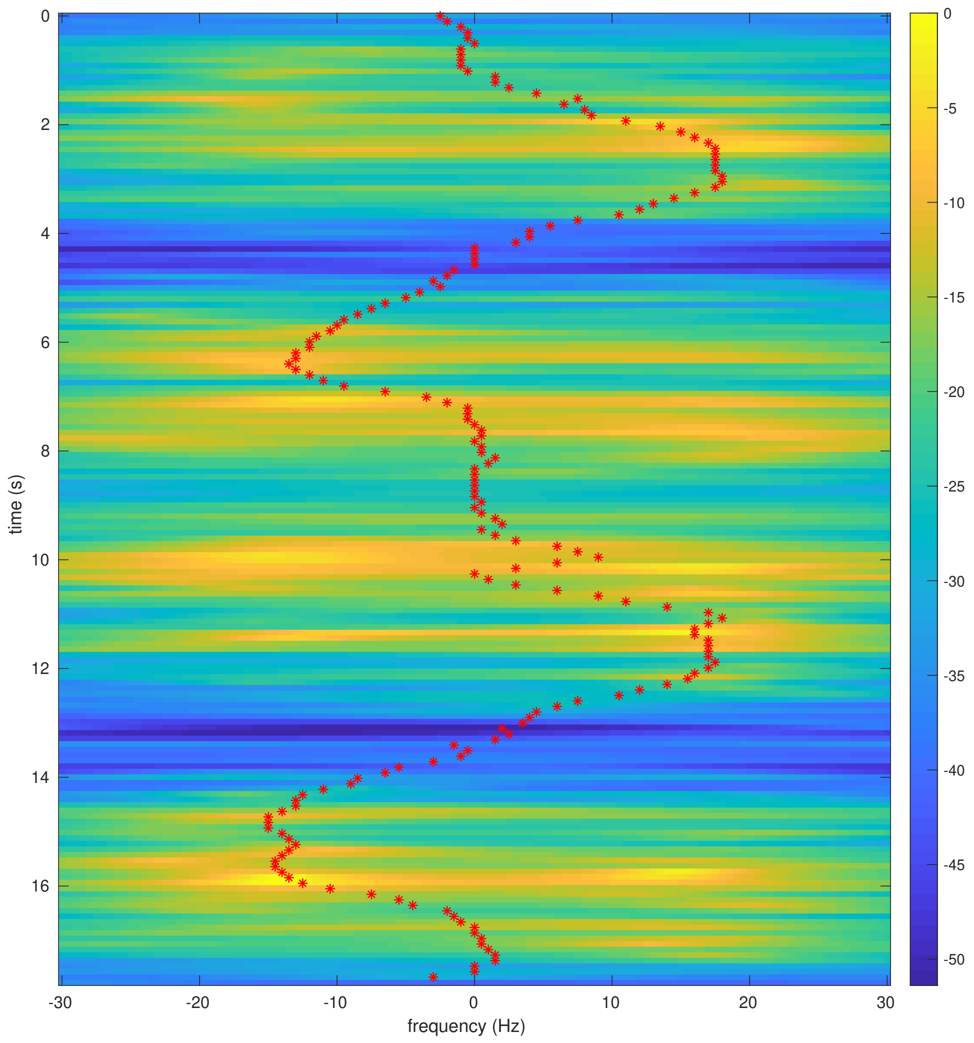


Fig. 5.43: Test 4 with the optimal path plotted on a MUSIC spectrum, NC antenna.

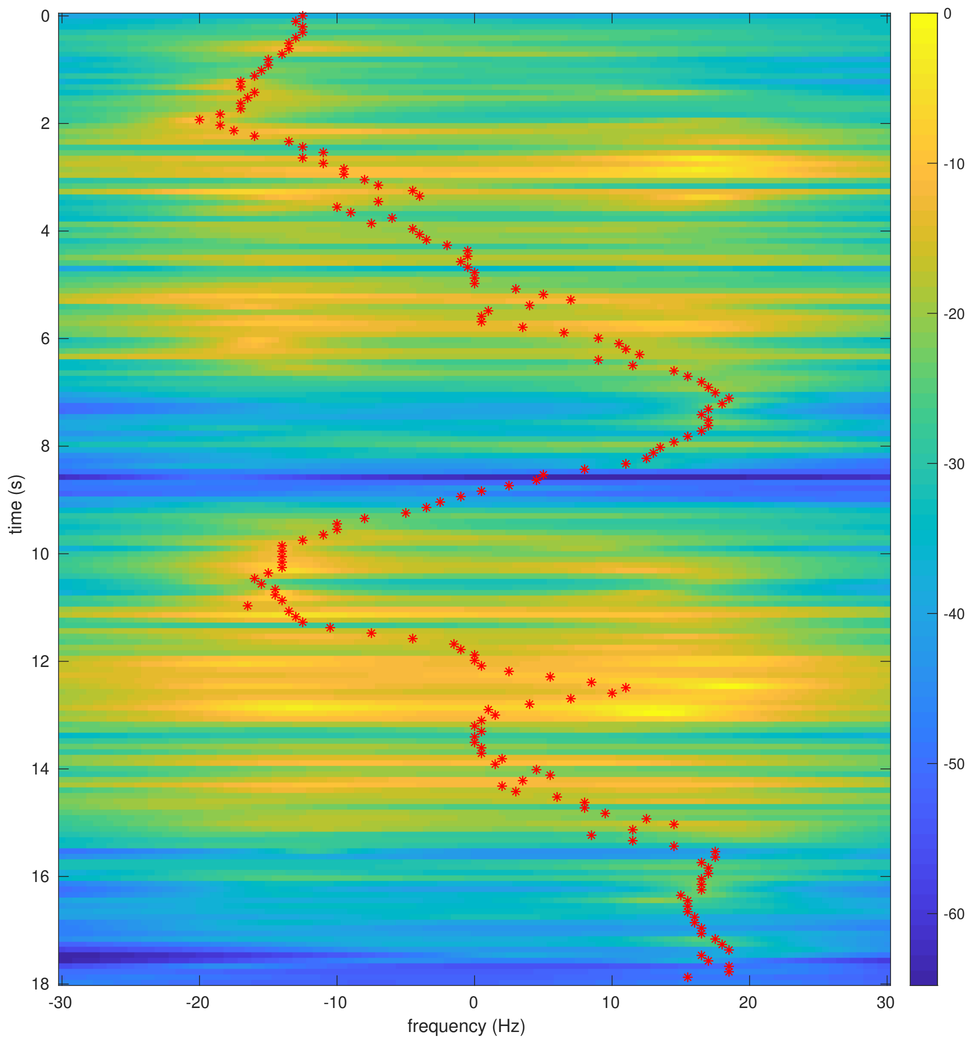


Fig. 5.44: Test 0 with the optimal path plotted on a MUSIC spectrum, NW antenna.

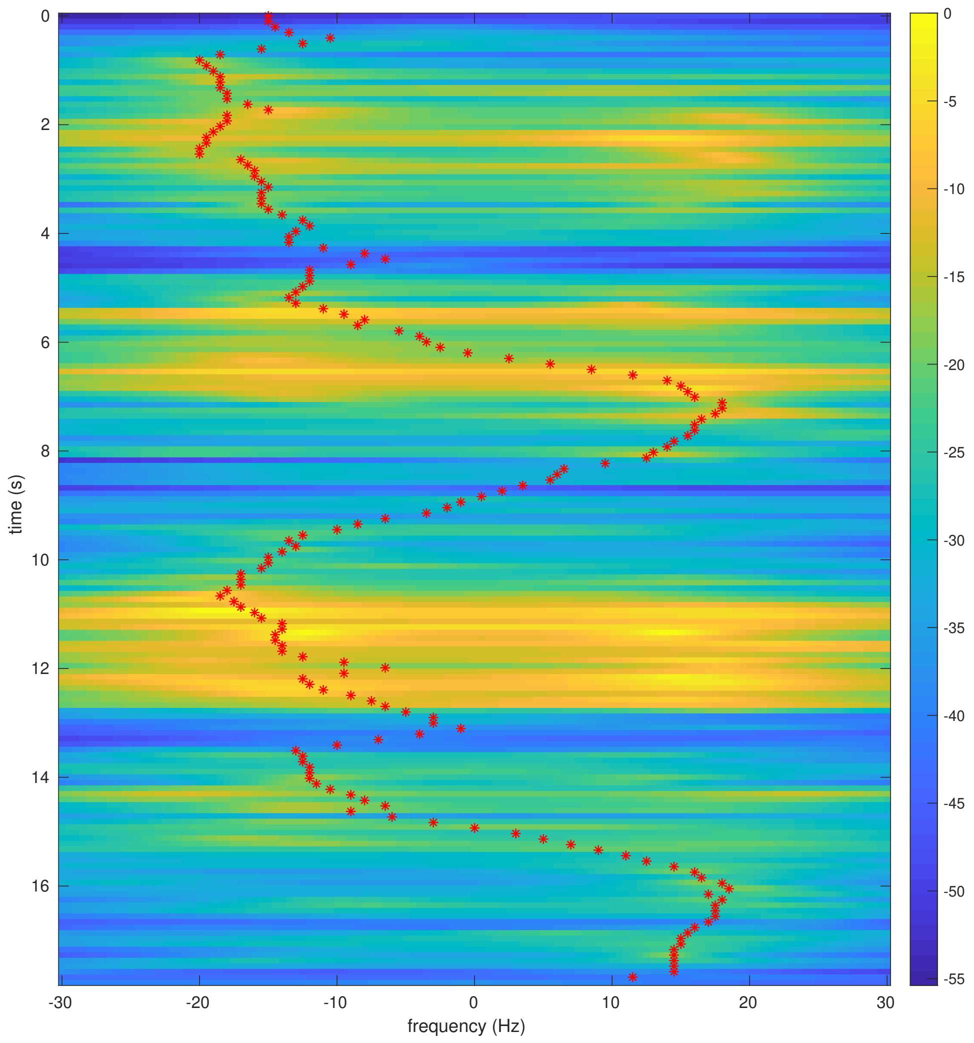


Fig. 5.45: Test 4 with the optimal path plotted on a MUSIC spectrum, NW antenna.

5.3 Discussion

The results of these tests show it is possible to extract the Doppler frequency path from a communication signal without needing a reference signal. The hardware of the system limits the applications this system can perform in, but additional tuning of the system could result in improved results for the less ideal hardware. However, finding a better tuning of all of the parameters is difficult.

In order to best set up and tune the whole system, it is best to split the system into four parts and tune in the following order: the communication system setup, the carrier frequency offset removal tuning, the spectrum estimation tuning, and the path extraction tuning. Each of these pieces can be tuned separately, but it is important to remember that each part significantly affects all the parts following it.

5.3.1 Communication System Setup

Tuning the communication system is of key importance to gathering useful Doppler echo data. The most important piece of this tuning is ensuring the system locks on to the signal constellation with the least noise on the constellation as possible.

The transmitter and receiver devices directly influence the amount of noise on the signal. The best devices are those without much noise or drift on their clocks. The less the carrier frequency offset drifts, the easier it is for the Second DEM to estimate and demodulate this frequency offset from the extracted Doppler signal.

When building the Doppler system, it is also important to keep in mind the placement of the antennas. The results suggest the further apart the receiver, transmitter, and target are from each other, the weaker the signal is. This means it is important to keep all of these as close together as possible. If the application requires them to be far apart, then using an amplifier on the transmitter may improve the results.

The parameters in GNU Radio were best set through trial and error. Since GNU Radio runs in real-time, adjusting these parameters until the signal locks on was done in real-time before collecting the data. Once the correct set of parameters are found, there is typically little need to change them from test to test. This resulted in the parameters of the communication system changing only slightly from test to test.

5.3.2 Carrier Frequency Offset Removal Tuning

Removing the carrier frequency offset is an important step in extracting the Doppler using the Second DEM. Difficulty in removing the carrier frequency offset results in a noisier spectrogram. The key to removing the carrier frequency offset is finding the proper parameters to get an accurate estimate of the carrier frequency offset without estimating the Doppler frequency.

This becomes a trade-off problem. If the parameters are set to aggressively track the carrier frequency offset, then the Doppler frequency may negatively affect this tracking and result in the spectrogram becoming noisier. This occurs when the path extraction uses the maximum value algorithm or the Viterbi algorithm with a low K value. However, if the parameters are set to slowly track the carrier frequency offset, then the algorithm may not track all of the carrier frequency offset's changes and also result in a noisy spectrogram. This occurs when the path extraction uses the Viterbi algorithm with a high K value.

This is the only situation where a strong Doppler echo negatively affects the system. Usually, it is better to have a stronger Doppler echo, but this can become a problem when estimating the carrier frequency offset. In the situation where the carrier frequency offset is extra noisy, the stronger Doppler signal may negatively affect the system as a whole. Overall, it is best to make sure the carrier frequency offset is the least noisy it can be to make it easier to track despite a strong Doppler echo interfering with the carrier frequency extraction.

5.3.3 Spectrum Estimation Parameter Tuning

The parameters for the spectrum estimation are the hardest part of the system to tune. Each parameter in the spectrum estimation subtly influences the optimal path parameters. Often, this results in adjusting the spectrum estimation parameters at the same time as the path extraction parameters.

The MUSIC algorithm is the preferred method used among all the tests due to the ability to improve spectral resolution. All but six of the spectrograms in this chapter use this method, as it often resulted in a better path. However, the DFT was often used while generating these results to help determine the best parameters for the MUSIC algorithm.

The MUSIC algorithm works well when it is properly tuned. When MUSIC is not properly tuned, it can easily give erroneous results. It is hard to determine from looking at a MUSIC spectrogram if the results are poor, as in some of the later tests it became difficult to determine the Doppler path from simply looking at the spectrogram. The DFT is a better algorithm for seeing what is going on exactly, even with less resolution. Therefore, it is often better to compare the spectrograms from both algorithms side by side when tuning the MUSIC algorithm.

5.3.4 Doppler Path Extraction Parameter Tuning

Tuning the parameters for the Doppler path extraction is more straightforward than tuning the parameters for the spectrum estimation; it is easier to see the change each parameter makes, but it can be a little overwhelming given how many different parameters there are. There are nine parameters to adjust when using the BCJR algorithm with the balance factor adjustment. However, the Viterbi parameters K and δ can often be set to the same value as the BCJR parameters K_1 and δ. This is because these parameters control essentially the same things in both algorithms. Both K and K_1 control the weight on the branch metric, while δ in both algorithms controls the allowed state transitions.

Situations may arise where it is advantageous to give one algorithm more freedom to allow transitions while keeping the other more restrictive, but usually it is better to keep them at the same value. BCJR makes some improvements upon Viterbi, but in many ways it is redundant.

To also simplify the number of parameters, it is useful to set either K_1 or K_2 and then only adjust the other. This is because the ratio of these weights is more important than the magnitude of these weights. Therefore, by making one of these constant and changing the other, the best path can more easily be found.

Most of the balance factor parameters can usually be set and left alone. The parameters σ, Λ, and K_3 all control the shape of the artificial spectrum. These parameters can often take some trial and error to get correct. K_0 changes how much weight is given to the balance factor and can also be set through trial and error.

ϵ is the most important parameter for the balance factor, as it determines when to apply the effects. The key is to apply the balance factor when it is most probable for the Doppler to be near 0 Hz. This occurs when the balance factor is close to 0. However, this can change easily from test to test depending on many factors. Therefore, this parameter is the first to change when there appears to be problems with the path not going to DC when it should.

CHAPTER 6
CONCLUSION

The purpose of this thesis was to explore a new method for extracting the Doppler frequencies from a communication signal. Traditionally, the Doppler frequencies are extracted using the CAF function, which requires the separation of the Doppler echo and the direct path signal [12]. This research studied a new way of performing the Doppler extraction to avoid the need for this separation. This new method takes advantage of the communication signal structure.

The First DEM was tested using a GNU Radio communication system. The results revealed that this method only worked when the transmitter and receiver had synchronized clocks. However, the Second DEM was developed for the case when the transmitter and receiver did not have synchronized clocks. This allowed the system to work in the more realistic case.

GNU Radio used the First DEM and Second DEM to extract the Doppler signal, but extracting the Doppler frequency path from this signal was difficult due to the low SIR and SNR of the signal. Additional processing was developed and implemented in MATLAB in order to improve the Doppler frequency path extraction. The key processing techniques used were the MUSIC algorithm, the Viterbi algorithm, and a balance factor adjusted BCJR algorithm.

The MUSIC algorithm was able to produce a higher resolution frequency spectrum by removing some of the noise on the signal. The Viterbi algorithm was able to find a smooth continuous path of highest likelihood through the Doppler echo spectrogram. The balance factor adjusted BCJR algorithm was able to make up for the loss of the Doppler path when the Doppler echo was at 0 Hz. Together, these algorithms were able to successfully extract the expected Doppler frequency path, as shown in Chapter 5.

This thesis also explored the limitations of this system. This system is limited by the strength of the Doppler echo and the phase noise on the carrier frequency offset. Ensuring the system has good receivers and limiting the size of the tracking area allows the system to overcome these limitations. This system also is difficult to tune because of the number of parameters for adjusting the algorithms in the system. More time optimizing the system could lead to improved results.

The purpose for extracting the Doppler echo is for indoor target tracking. Once the Doppler echo is extracted, a Kalman filter or a particle filter can use the Doppler echo to track the location of the moving target. The Doppler echo from multiple receivers is required for this tracking using either of the filters. This thesis explored the extraction of the Doppler frequency path with the goal of extracting this path at multiple receivers to use in the indoor tracking system. Implementing the tracking system as described in the paper by Chen, et al. using this Doppler data is the next step in the research [3].

Figure 6.1 shows an initial tracking plot using data from Test 0 of the February 12, 2021 test set. This test involved a person carrying an aluminum sheet jogging in between the SW and SC receivers twice. This plot was generated using an extended Kalman filter modified to correct the path from going outside of the marked hallways. The unfilled circle denotes the starting point of the person jogging up and down the hall while the star represents the ending point. This initial plot suggests promising results for future research using this new Doppler extraction system. Additional work can be done to improve the extending Kalman filter and overall indoor human tracking system.

There are also other algorithms and techniques to explore to improve this Doppler extraction system. The Second DEM was developed to overcome the carrier frequency offset problems the First DEM suffered from. However, there may be ways to overcome the problems of the First DEM. Section 4.4 explored one possibility, but more research into this method and other methods would be beneficial.

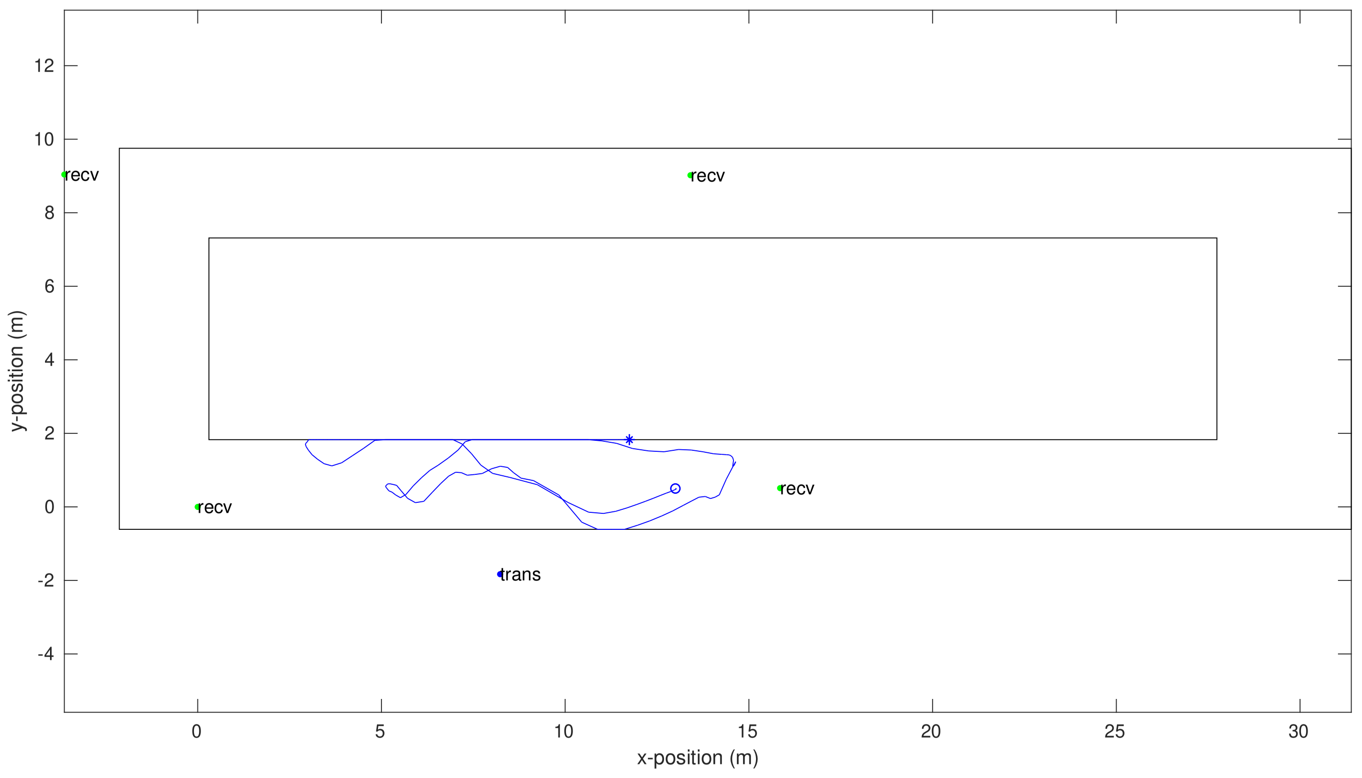


Fig. 6.1: Initial target tracking test using Doppler extracted data from the February 12, 2021 tests.

REFERENCES

[1] H. Griffiths, “Developments in bistatic and networked radar,” in *Proceedings of 2011 IEEE CIE International Conference on Radar*, vol. 1, 2011, pp. 10–13.

[2] K. Chetty, G. E. Smith, and K. Woodbridge, “Through-the-wall sensing of personnel using passive bistatic WiFi radar at standoff distances,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 4, pp. 1218–1226, 2012.

[3] Q. Chen, B. Tan, K. Woodbridge, and K. Chetty, “Indoor target tracking using high Doppler resolution passive Wi-Fi radar,” in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5565–5569.

[4] B. Tan, A. Burrows, R. Piechocki, I. Craddock, Q. Chen, K. Woodbridge, and K. Chetty, “Wi-Fi based passive human motion sensing for in-home healthcare applications,” in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, 2015, pp. 609–614.

[5] Y. Li, X. Wang, and Z. Ding, “Multi-target position and velocity estimation using OFDM communication signals,” *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1160–1174, 2020.

[6] Chenqi Zhang, Yong Wu, J. Wang, and Zhen Luo, “FM-based multi-frequency passive radar system,” in *2016 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, 2016, pp. 1–4.

[7] P. Zhang, Y. Wu, J. Wang, and J. Qiao, “Real-time signal processing for FM-based passive bistatic radar using GPUs,” in *2014 19th International Conference on Digital Signal Processing*, 2014, pp. 536–540.

[8] P. E. Howland, D. Maksimiuk, and G. Reitsma, “FM radio based bistatic radar,” *IEEE Proceedings - Radar, Sonar and Navigation*, vol. 152, no. 3, pp. 107–115, 2005.

[9] M. Edrich, A. Schroeder, and F. Meyer, “Design and performance evaluation of a mature FM/DAB/DVB-T multi-illuminator passive radar system,” *IET Radar, Sonar Navigation*, vol. 8, no. 2, pp. 114–122, 2014.

[10] W. Li, B. Tan, and R. Piechocki, “Opportunistic Doppler-only indoor localization via passive radar,” in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, 2018, pp. 467–473.

[11] I. Milani, F. Colone, C. Bongioanni, and P. Lombardo, “WiFi emission-based vs passive radar localization of human targets,” in *2018 IEEE Radar Conference (RadarConf18)*, 2018, pp. 1311–1316.

[12] B. Tan, K. Woodbridge, and K. Chetty, "A real-time high resolution passive WiFi Doppler-radar and its applications," in *2014 International Radar Conference*, 2014, pp. 1–6.

[13] L. Schauer, P. Marcus, and C. Linnhoff-Popien, "Towards feasible Wi-Fi based indoor tracking systems using probabilistic methods," in *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, 2016, pp. 1–8.

[14] Y. Lai, C. Chou, M. Tang, T. Horng, and F. Wang, "Finger gesture sensing and recognition using a Wi-Fi-based passive radar," in *2019 IEEE MTT-S International Microwave Symposium (IMS)*, 2019, pp. 293–296.

[15] M. Rice, *Digital Communications: A Discrete-Time Approach.* Self-published, 2018.

[16] N. J. Willis, *Bistatic Radar.* Artech House, 1991.

[17] R. W. S. Oppenheim, Alan V., *Discrete-Time Signal Processing.* Pearson Higher Education, Inc., 2010.

[18] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing.* Prentice Hall, 2000.

[19] T. K. Moon, *Error Correction Coding.* John Wiley & Sons, Inc., 2005.

APPENDICES

APPENDIX A

Test Parameters

The following pages contain the spectrum generation and path extraction parameters used to generate the figures in Chapter 5. These parameters are kept in three separate tables.

The first table, Table A.1, lists all the parameters needed for the carrier frequency offset removal. Only tests using the Second DEM need to remove the carrier frequency offset. The parameters for this removal are the type of spectrogram generated (DFT or MUSIC), the step size between each line of the spectrogram, the length of the data frame for each line of the spectrogram, and the resolution of the spectrogram. In addition, if MUSIC was used the P and M values are recorded, and if the Viterbi algorithm was used for the path extraction its parameters are also recorded.

The second table, Table A.2, lists all the parameters needed for the Doppler signal spectrum estimation. The parameters for this generation are the type of spectrogram generated (DFT or MUSIC), the step size between each line of the spectrogram, the length of the data frame for each line of the spectrogram, and the resolution of the spectrogram. In addition, if MUSIC was used the P and M values are recorded.

The third table, Table A.3, lists all the parameters needed for the Doppler frequency path extraction. It lists the algorithms used to extract the path and any parameters for the algorithms.

All three tables list the Doppler extraction method for the test. Any unused parameters in the tables will list a n/a to note it was not used.

Table A.1: Carrier frequency offset removal parameters

Figure	Method	Type	Step Length (s)	Frame Length (s)	Resolution (Hz)	P/M	Path	K	δ
5.3	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.4	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.5	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.6	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.7	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.8	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.9	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.10	1	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.11	2	DFT	0.1	0.2	0.1	n/a	max	n/a	n/a
5.12	2	MUSIC	0.1	0.2	0.1	1/5	max	n/a	n/a
5.13	2	MUSIC	0.1	0.2	0.1	1/5	max	n/a	n/a
5.14	2	DFT	0.1	0.2	0.1	n/a	Viterbi	0	0.5
5.15	2	DFT	0.1	0.2	0.1	n/a	Viterbi	0	0.5
5.20	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	20	0.5

continued on next page

Table A.1: Carrier frequency offset removal parameters - *continued from previous page*

Figure	Method	Type	Step Length (s)	Frame Length (s)	Resolution (Hz)	P/M	Path	K	δ
5.21	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	20	0.5
5.22	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.23	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.24	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.25	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.26	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.27	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.28	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.29	2	MUSIC	0.1	0.2	0.1	1/5	Viterbi	30	0.5
5.30	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5
5.31	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5
5.32	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5
5.33	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5
5.34	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5

continued on next page

Table A.1: Carrier frequency offset removal parameters - *continued from previous page*

Figure	Method	Type	Step Length (s)	Frame Length (s)	Resolution (Hz)	P/M	Path	K	δ
5.35	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5
5.36	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5
5.37	2	MUSIC	0.1	0.2	0.1	1/6	Viterbi	50	0.5
5.38	2	MUSIC	0.05	0.05	0.1	1/5	Viterbi	30	0.25
5.40	2	MUSIC	0.05	0.05	0.1	1/5	Viterbi	30	0.25
5.42	2	MUSIC	0.05	0.05	0.1	1/5	Viterbi	30	0.25
5.44	2	MUSIC	0.05	0.05	0.1	1/5	Viterbi	30	0.25

Table A.2: Spectrum estimation parameters

Figure	Method	Type	Step Length (s)	Frame Length (s)	Resolution (Hz)	P/M
5.3	1	DFT	0.1	0.2	0.1	n/a
5.4	1	DFT	0.1	0.2	0.1	n/a
5.5	1	DFT	0.1	0.2	0.1	n/a
5.6	1	DFT	0.1	0.2	0.1	n/a
5.7	1	DFT	0.1	0.2	0.1	n/a
5.8	1	MUSIC	0.1	0.2	0.1	50/200
5.9	1	MUSIC	0.1	0.2	0.1	50/200
5.10	1	MUSIC	0.1	0.2	0.1	50/200
5.11	2	MUSIC	0.1	0.2	0.1	50/200
5.12	2	MUSIC	0.1	0.2	0.1	70/200
5.13	2	MUSIC	0.1	0.2	0.1	70/200
5.14	2	MUSIC	0.1	0.2	0.1	70/200
5.15	2	MUSIC	0.1	0.2	0.1	70/200
5.20	2	MUSIC	0.1	0.2	0.1	70/250
5.21	2	MUSIC	0.1	0.2	0.1	70/250
5.22	2	MUSIC	0.1	0.2	0.1	70/200
5.23	2	MUSIC	0.1	0.2	0.1	70/200
5.24	2	MUSIC	0.1	0.2	0.1	70/200
5.25	2	MUSIC	0.1	0.2	0.1	70/200
5.26	2	MUSIC	0.1	0.2	0.1	70/200
5.27	2	MUSIC	0.1	0.2	0.1	70/200
5.28	2	MUSIC	0.1	0.2	0.1	70/200
5.29	2	MUSIC	0.1	0.2	0.1	70/200

continued on next page

Table A.2: Spectrum estimation parameters - *continued from previous page*

Figure	Method	Type	Step Length (s)	Frame Length (s)	Resolution (Hz)	P/M
5.30	2	MUSIC	0.1	0.2	0.1	70/200
5.31	2	MUSIC	0.1	0.2	0.1	70/200
5.32	2	MUSIC	0.1	0.2	0.1	70/200
5.33	2	MUSIC	0.1	0.2	0.1	70/200
5.34	2	MUSIC	0.1	0.2	0.1	70/200
5.35	2	MUSIC	0.1	0.2	0.1	70/200
5.36	2	MUSIC	0.1	0.2	0.1	70/200
5.37	2	MUSIC	0.1	0.2	0.1	70/200
5.38	2	MUSIC	0.1	0.2	0.5	70/250
5.40	2	MUSIC	0.1	0.2	0.5	70/250
5.42	2	MUSIC	0.1	0.2	0.5	70/250
5.44	2	MUSIC	0.1	0.2	0.5	70/250

Table A.3: Path extraction parameters

Figure	Method	Path	Viterbi		BCJR			BCJR, BF				
			K	δ	K_1	K_2	δ	K_0	K_3	σ	Λ	ϵ
5.3	1	Viterbi	0.2	3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.4	1	Viterbi	0.2	3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.5	1	Viterbi	0.2	3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.6	1	Viterbi	0.5	3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.7	1	Viterbi	0.5	3	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.8	1	Viterbi	0.8	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.9	1	Viterbi	0.8	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.10	1	Viterbi	0.8	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.11	2	Viterbi	0.5	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.12	2	Viterbi	1.0	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.13	2	Viterbi	1.0	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.14	2	Viterbi	0.8	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.15	2	Viterbi	0.8	2	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
5.20	2	BCJR	0	2	0.5	0.5	2	n/a	n/a	n/a	n/a	n/a
5.21	2	BCJR	0	2	0.5	0.5	2	n/a	n/a	n/a	n/a	n/a

continued on next page

Table A.3: Path extraction parameters - *continued from previous page*

Figure	Method	Path	Viterbi		BCJR			BCJR, BF				
			K	δ	K_1	K_2	δ	K_0	K_3	σ	Λ	ϵ
5.22	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.23	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.24	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.25	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.26	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.27	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.28	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.29	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.07
5.30	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09
5.31	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09
5.32	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09
5.33	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09
5.34	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09
5.35	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09
5.36	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09

continued on next page

Table A.3: Path extraction parameters - *continued from previous page*

Figure	Method	Path	Viterbi		BCJR			BCJR, BF				
			K	δ	K_1	K_2	δ	K_0	K_3	σ	Λ	ϵ
5.37	2	BCJR,BF	0.1	2	0.5	0.5	2	2	0.25	20	5	0.09
5.38	2	BCJR,BF	0.0	3	0.9	0.5	3	2	0.25	20	5	0.07
5.40	2	BCJR,BF	0.0	3	0.9	0.5	3	2	0.25	20	5	0.07
5.42	2	BCJR,BF	0.0	3	0.9	0.5	3	2	0.25	20	5	0.07
5.44	2	BCJR,BF	0.0	3	0.9	0.5	3	2	0.25	20	5	0.07

APPENDIX B

Derivation of Path Derivative

This appendix uses geometry to derive the formula for the derivative of the path length

$$\frac{dR}{dt} = \lim_{\Delta t \to 0} \frac{R(t + \Delta t) - R(t)}{\Delta t} \tag{B.1}$$

in the situation shown in Figure B.1.

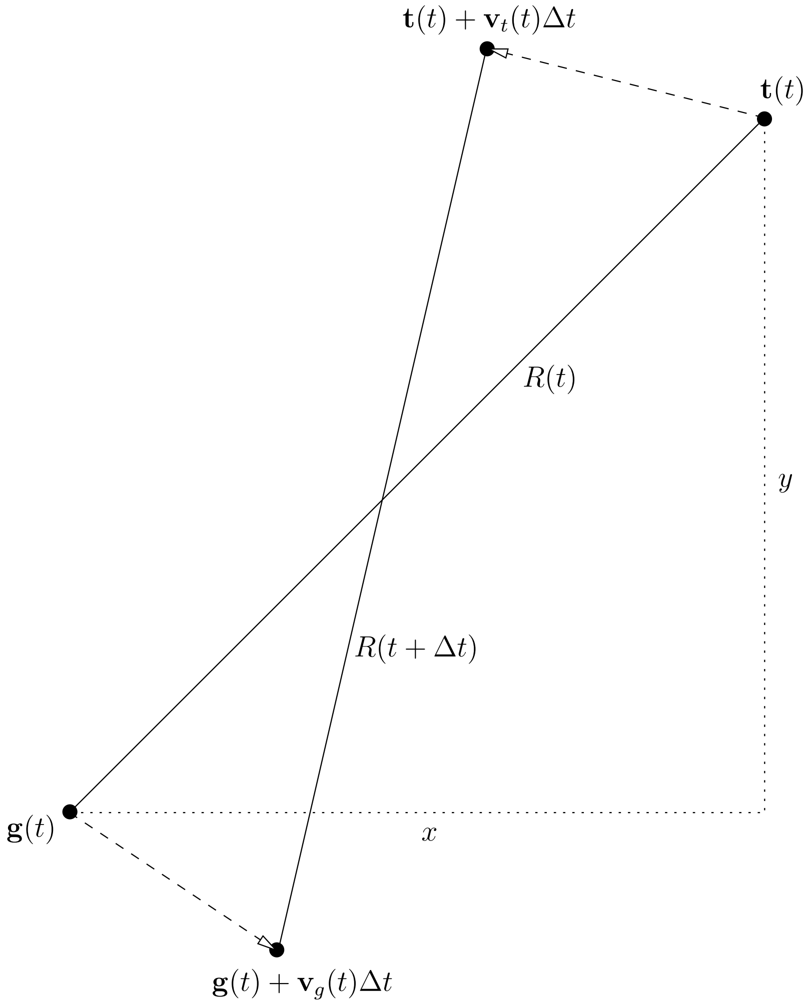

$\mathbf{t}(t) + \mathbf{v}_t(t)\Delta t$
$\mathbf{t}(t)$
$R(t)$
y
$R(t + \Delta t)$
$\mathbf{g}(t)$
x
$\mathbf{g}(t) + \mathbf{v}_g(t)\Delta t$

Fig. B.1: Diagram of a changing path length between two moving objects.

With the position of one moving object defined as $\mathbf{t}(t) = (t_x, t_y)$ with velocity $\mathbf{v}_t(t) = (v_{xt}, v_{yt})$, and the position of the other moving object defined as $\mathbf{g}(t) = (g_x, g_y)$ with velocity $\mathbf{v}_g(t) = (v_{xg}, v_{yg})$, we find

$$\begin{aligned}
\frac{R(t+\Delta t) - R(t)}{\Delta t} &= \frac{\Big|\Big|\big(\mathbf{t}(t) + \mathbf{v_t}(t)\Delta t\big) - \big(\mathbf{g}(t) + \mathbf{v_g}(t)\Delta t\big)\Big|\Big| - \Big|\Big|\mathbf{t}(t) - \mathbf{g}(t)\Big|\Big|}{\Delta t} \\
&= \frac{\sqrt{\Big((t_x + v_{xt}\Delta t) - (g_x + v_{xg}\Delta t)\Big)^2 + \Big((t_y + v_{yt}\Delta t) - (g_y + v_{yg}\Delta t)\Big)^2}}{\Delta t} \\
&\quad - \frac{\sqrt{\Big(t_x - g_x\Big)^2 + \Big((t_y - g_y\Big)^2}}{\Delta t} \\
&= \frac{\sqrt{\Big((t_x - g_x) + \Delta t(v_{xt} - v_{xg})\Big)^2 + \Big((t_y - g_y) + \Delta t(v_{yt} - v_{yg})\Big)^2}}{\Delta t} \\
&\quad - \frac{\sqrt{\Big(t_x - g_x\Big)^2 + \Big((t_y - g_y\Big)^2}}{\Delta t}.
\end{aligned}$$

Defining $x = t_x - g_x$, $y = t_y - g_y$, $\Delta v_x = v_{xt} - v_{xg}$, and $\Delta v_y = v_{yt} - v_{yg}$ leads to

$$\begin{aligned}
\frac{R(t+\Delta t) - R(t)}{\Delta t} &= \frac{\sqrt{\Big(x + \Delta t(\Delta v_x)\Big)^2 + \Big(y + \Delta t(\Delta v_y)\Big)^2} - \sqrt{x^2 + y^2}}{\Delta t} \\
&= \frac{\sqrt{x^2 + 2x\Delta t(\Delta v_x) + \big(\Delta t(\Delta v_x)\big)^2 + y^2 + 2y\Delta t(\Delta v_y) + \big(\Delta t(\Delta v_y)\big)^2} - \sqrt{x^2 + y^2}}{\Delta t}.
\end{aligned}$$

The terms $(\Delta t(\Delta v_x))^2$ and $(\Delta t(\Delta v_y))^2$ can be dropped as they will be insignificant in the limit as $\Delta t \to 0$,

$$\begin{aligned}
\frac{R(t+\Delta t) - R(t)}{\Delta t} &\approx \frac{\sqrt{x^2 + 2x\Delta t(\Delta v_x) + y^2 + 2y\Delta t(\Delta v_y)} - \sqrt{x^2 + y^2}}{\Delta t} \\
&= \frac{\sqrt{\big(x^2 + y^2\big) + 2\Delta t\big(x(\Delta v_x) + y(\Delta v_y)\big)} - \sqrt{x^2 + y^2}}{\Delta t} \\
&= \frac{\sqrt{x^2 + y^2}\sqrt{1 + \frac{2\Delta t\big(x(\Delta v_x) + y(\Delta v_y)\big)}{x^2 + y^2}} - \sqrt{x^2 + y^2}}{\Delta t}.
\end{aligned}$$

Using the identity $\sqrt{1+\alpha} \approx 1 + \alpha/2$ when $\alpha \ll 1$, the equation simplifies to

$$\begin{aligned}
\frac{R(t+\Delta t) - R(t)}{\Delta t} &\approx \frac{\sqrt{x^2+y^2}\left(1 + \frac{\Delta t\big(x(\Delta v_x)+y(\Delta v_y)\big)}{x^2+y^2}\right) - \sqrt{x^2+y^2}}{\Delta t} \\
&= \frac{\sqrt{x^2+y^2} + \frac{\Delta t\big(x(\Delta v_x)+y(\Delta v_y)\big)}{\sqrt{x^2+y^2}} - \sqrt{x^2+y^2}}{\Delta t} \\
&= \frac{\Delta t\big(x(\Delta v_x) + y(\Delta v_y)\big)}{\Delta t\sqrt{x^2+y^2}} \\
&= \frac{x(\Delta v_x) + y(\Delta v_y)}{\sqrt{x^2+y^2}}.
\end{aligned}$$

Plugging back in the values for x, y, Δv_x, and Δv_y, and taking the limit leads to

$$\frac{dR}{dt} = \frac{(t_x - g_x)(v_{xt} - v_{xg}) + (t_y - g_y)(v_{yt} - v_{yg})}{\sqrt{(t_x-g_x)^2 + (t_y-g_y)^2}} = \frac{(\mathbf{v}_t(t) - \mathbf{v}_g(t)) \cdot (\mathbf{t}(t) - \mathbf{g}(t))}{||\mathbf{t}(t) - \mathbf{g}(t)||}. \tag{B.2}$$

At this point it is recognized that

$$\frac{\mathbf{t}(t) - \mathbf{g}(t)}{||\mathbf{t}(t) - \mathbf{g}(t)||} = \hat{\mathbf{u}}(t)$$

is a unit vector pointing from $\mathbf{g}(t)$ to $\mathbf{t}(t)$. This allows (B.2) to be written as

$$\frac{dR}{dt} = (\mathbf{v}_t(t) - \mathbf{v}_g(t)) \cdot \frac{\mathbf{t}(t) - \mathbf{g}(t)}{||\mathbf{t}(t) - \mathbf{g}(t)||} = (\mathbf{v}_t(t) - \mathbf{v}_g(t)) \cdot \hat{\mathbf{u}}(t). \tag{B.3}$$

APPENDIX C

Code Listings

The MATLAB code used to generate the spectrograms throughout this thesis is provided in the following sections. This code is designed to use any of the methods discussed in this thesis.

Section C.1 lists the overhead files for the project. There are four overhead files for the system. These files have been split up to make it easier to control the system. Listing C.1 controls the parameters for the entire system. This file was the main file edited in adjusting the system. Listing C.2 is the overhead file for performing the spectrum estimation, and Listing C.3 is the overhead file for performing the path extraction. Listing C.4 is the overhead file responsible for plotting the results. This section also contains a function for reading in the data in Listing C.5.

Section C.2 contains the functions for generating the DFT and MUSIC spectra. Listing C.6 and Listing C.7 contain functions for the DFT. Listing C.8 and Listing C.9 contain functions for the MUSIC algorithm.

Section C.3 lists the various Viterbi functions for the different path extraction methods. The basic Viterbi algorithm used is in Listing C.10. The Viterbi algorithms for the symmetry correction of Section 4.4 are in Listing C.11 and Listing C.12.

Section C.4 lists the BCJR functions for the different path extraction methods. Listing C.13 contains the basic BCJR function, while Listing C.14 contains the balance factor adjusted BCJR algorithm. This section also includes the balance factor function in Listing C.15.

C.1 Overhead Files

Listing C.1: Overhead code for adjusting all of the parameters and controlling the entire system

```
%%%%%%%%%%%%%%%%%%%%%
%%% SET PARAMETERS %%%
%%%%%%%%%%%%%%%%%%%%%
%%% Control Variables %%%
% Whether to limit the length of the data
Data_limit = 0;
% This turns off the Spectrum generation when working with the same data
% for quiker results
Path_gen_only = 0;
% This shows the balance factor on plot as either above threshold or not
show_BF = 0;
% Which Doppler method to use (1 or 2)
Doppler_method = 2;
% Spectrum for output (DFT or MUSIC)
Spectrum = 'MUSIC';
% Method to find optimal path
% (none, viterbi, max, symm_viterbi, BCJR, or BCJR_BF)
Path = 'BCJR_BF';
% (METHOD1 only)
% Where to get the data for the sign extraction (PLL or SS)
Sign_ext = 'SS';
% (METHOD2 only)
% Method to find carrier frequency offset (DFT or MUSIC)
C_spec = 'MUSIC';

%%% Choose test %%%
% Date of the data set
date = "21_02_12";
% Which test to run from the data set
test_num_l = 0;
% Which receiver if multiple (type "none" if only one receiver)
```

```
receiver_l = "SW";

%%% Figure numbers %%%
time_plt = 0;
carr_offset_plt = 1;
DFT_out_plt = 3;
MUSIC_out_plt = 4;
BCJR_plot = 5;

%%% Spectral Estimation Variables %%%
% Symbol Sample rate
Fs = 153600/8;
% Center frequency
cenF = 0;
% Maximum frequency extension in either +/- direction
maxF = 30;
% Spectrum resolution
delF = 0.5;
% Data frame length, in seconds
DL = 0.2;
% Data time step
D_step = 0.1;
% (MUSIC only)
% Estimate of the number of signals
P = 70;
% M improves the resolution of MUSIC, at expense of computations
M = 250;
% (Data limit)
% When to start the data (time in seconds)
D_start = 5;
% When to end data (time in seconds)
D_end = 15;

%%% Path Extraction Variables %%%
%%% (Viterbi, normal or symmetric) %%%
```

```
% Maximum frequency shift from state to state for viterbi
delta_v = 2;
% Branch metric weight
K = 1;
%%% (Viterbi Sign) %%%
% Branch metric weight
K_sign = 0.01;
% How much data should be used to find the sign
sign_DL = DL;
%%% (BCJR) %%%
% Maximum frequency shift from state to state for BCJR
delta_BCJR = 3;
% Weight for the branch metric
K1 = 0.9;
% Weight for the state value
K2 = 0.5;
%%% (BCJR BF) %%%
% Threshold to turn on adjustment spectrum
epsilon = 0.07;
% Weight given to balance factor
K0 = 2;
% Width of adjustment spectrum
sigma = 20;
% Linear drop rate away from 0 Hz
K3 = 0.25;
% Maximum value of adjustment spectrum, located at 0 Hz
Lambda = 5;
%%% (For plotting balance factor) %%%
% Position on spectrogram
graph_pos = -25;
% Scaling variable to make changes in BF noticable
dX = 10;

%%% (METHOD2) Carrier spectrum setup %%%
% Maximum frequency MUSIC extension in either +/- direction
```

```
maxF_i = 3;
% Spectrum resolution
delF_i = 0.1;
% Data frame length, in seconds
DL_i = 0.05;
% How much time to step forward for each data point
D_step_i = 0.05;
% How many signals you should find in data
P_i = 1;
% M improves the resolution of MUSIC, at expense of computations
M_i = 5;
%%% Viterbi parameters %%%
% Branch metric weight
K_i = 30;
% Furthest allowed state jump
delta_i = 0.25;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% EXECUTE THE EXTRACTION %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Cycle through all the tests
for test_num = test_num_l
    % Antenna number for outputing Doppler paths
    antenna_num = 1;
    clear dopp_mat;
    % Cycle through all the receivers in this test
    for ii = 1:length(receiver_l)
        receiver = receiver_l(ii);
        %Run the program and time it
        tic
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%% SPECTRUM ESTIMATION %%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        if(Path_gen_only == 0)
            doppler_spectrum_generation;
        end
        %%%%%%%%%%%%%%%%%%%%%%%%
        %%% PATH EXTRACTION %%%
        %%%%%%%%%%%%%%%%%%%%%%%%
        doppler_path_extraction;

        % Report time
        elapsedTime = toc;
        disp(newline);
        disp("Program took "+elapsedTime+" seconds to complete");
        disp(newline);

        %%%%%%%%%%%%%%%%%%%
        %%% MAKE PLOTS %%%
        %%%%%%%%%%%%%%%%%%%
        doppler_plots;

        % Update plot numbers so you don't lose them all for multiple runs
        carr_offset_plt = carr_offset_plt+1;
        DFT_out_plt = DFT_out_plt+1;
        MUSIC_out_plt = MUSIC_out_plt+1;
        BCJR_plot = BCJR_plot+1;

%         %%% Output the plots to files %%%
%         spec_path = output_plot_path+"/"+date+"_test"+test_num+"_"+receiver
    +"_"+Spectrum+"spec.png";
%         %prob_path = output_plot_path+"/"+date+"_test"+test_num+"_"+receiver
    +"_"+Spectrum+"prob.png";
%         saveas(spec,spec_path);
%         %saveas(probs,prob_path);

%         %%% Save path in matrix and move on to next antenna %%%
%         dopp_mat(antenna_num,:) = best_path;
%         % Update antenna number
%         antenna_num = antenna_num + 1;
```

```
    end
%     %%% Output Doppler matrix to a file for these tests %%%
%     output_file = output_data_path+"/"+date+"_test"+test_num+"_"+Spectrum+".
    mat";
%     save(output_file, 'dopp_mat', 'receiver_l');
end
```

Listing C.2: Overhead code to perform the spectrum estimation

```
% This file performs the spectrum estimation and produces a spectrogram
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% GET NEEDED DATA FIRST %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Determine if setup is correct or throw error %%%
if(Doppler_method == 2)
    if(strcmp(Path,'symm_viterbi'))
        error('Method 2 does not support the symmertic viterbi algorithm');
    end
else
    error('Please choose a correct Doppler method of 1 or 2');
end

%%% Determine file name %%%
data_path = "~/Documents/research_2020-21/data/";
if(receiver == "none")
    data_set = date + "_data/";
else
    data_set = date + "_data/"+receiver+"/";
end
% Generic file location
file_start = data_path+data_set+"test_"+test_num;
% Get the Doppler signal collected from method 1
file_dopp = file_start+"_doppler.bin";
% Get the Doppler signal collected form method 2
file_new_dopp = file_start+"_new_doppler.bin";
% Get zk from the carrier frequency synchronization block
file_PLL = file_start+"_carr_phase.bin";
% Get sk from the symbol synchronization block
file_sym_sync = file_start+"_sym_sync.bin";
% Get ak fromt he symbol decision block
file_sym_dec = file_start+"_sym_decision.bin";

%%% Read in the data %%%
```

```
if(Doppler_method == 1)
    d = get_complex_data(file_dopp);
    % Remove the mean from the data
    d = d - mean(d);
    len = length(d);
elseif(Doppler_method == 2)
    dn = get_complex_data(file_new_dopp);
    len = length(dn);
end
% Read in extra data for the symmetry correction, if needed
if(strcmp(Path,'symm_viterbi'))
    ss = get_complex_data(file_sym_sync);
    ahat = get_complex_data(file_sym_dec);
    if(strcmp(Sign_ext,'PLL'))
        PLL = get_complex_data(file_PLL);
    elseif(strcmp(Sign_ext,'SS'))
        ss = get_complex_data(file_sym_sync);
    end
end

%%% Limit the data length, if needed %%%%
if(Data_limit)
    % Compute the start and stop from the time given in seconds
    start = floor(D_start*Fs + 1);
    finish = floor(D_end*Fs);
    % Try block to catch error if requested data out of range
    try
        if(Doppler_method == 1)
            d = d(start:finish);
            len = length(d);
        elseif(Doppler_method == 2)
            dn = dn(start:finish);
            len = length(dn);
        end
        if(strcmp(Path,'symm_viterbi'))
```

```
            ss = ss(start:finish);
            ahat = ahat(start:finish);
        end
    catch ME
        msg_start = "Unable to start and stop data at given parameters";
        msg_end = "Data is "+len/Fs+" seconds long";
        msg = msg_start + newline + msg_end;
        error(msg);
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% DETERMINE OTHER VARIABLES %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Set up the data frames %%%
% How many data points in each frame
frame_length = floor(DL*Fs);
% How many frames to cycle through
n_points = floor((len-frame_length)/(Fs*D_step));
% How many points to advance each frame
dT = floor(Fs*D_step);

%%% Setup frequency bins %%%
% List of all frequency bins in continuous frequencies
F = -maxF+cenF:delF:maxF+cenF;
% Discrete frequency bins
f = F./Fs;

%%% (METHOD2) Setup frequency bins for finding carrier frequency offset %%%
if(Doppler_method == 2)
    % Number of data points in each frame
    frame_length_i = floor(DL_i*Fs);
    % How many frames to cycle through
    n_points_i = floor((len-frame_length_i)/(Fs*D_step_i));
    % How many points to advance each frame
    dT_i = floor(Fs*D_step_i);
    % To find carrier frequency offset location
    f_tot = -1/2:1/Fs:1/2;
end

%%% Set up for DFT and MUSIC algorithms %%%
if(strcmp(Spectrum,'DFT'))
    % Precompute DFT weights
    Wn = compute_Wn(f, frame_length);
elseif(strcmp(Spectrum,'MUSIC'))
    % Precompute the S vectors for MUSIC
    S = compute_S(f, M);
    % Precompute DFT weights for finding the balance factor
    Wn_cf = compute_Wn(f, frame_length);
else
    error('Only supported frequency spectrum algorithms are the DFT or MUSIC')
        ;
end
% Also set up for the carrier frequency offset spectrum estimation
if(Doppler_method == 2)
    if(strcmp(C_spec,'DFT'))
        % Precompute DFT weights
        Wn_tot = compute_Wn(f_tot, frame_length_i);
    elseif(strcmp(C_spec,'MUSIC'))
        % Precompute the S vectors for MUSIC
        S_tot = compute_S(f_tot, M_i);
    else
        error('Only supported frequency spectrum algorithms are the DFT or
            MUSIC');
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% (METHOD2) REMOVE CARRIER FREQUENCY OFFSET %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This method currently calculates the carrier frequency offset separately
% and removes it. This will not translate to real-time and will need to be
% integrated into other loops
if(Doppler_method == 2)
    % Create empty arrays to hold data
    y_mat = zeros(frame_length_i,n_points_i);

    %%% Find the frequency spectrum for the first time interval %%%
    % x is now the first frame of dn
    x = dn(1:frame_length_i);
    if(strcmp(C_spec,'DFT'))
        X_f_i = DFT(x, Wn_tot);
    elseif(strcmp(C_spec,'MUSIC'))
        X_f_i = MUSIC(x, P_i, M_i, S_tot);
    else
        error('Only supported frequency spectrum algorithms are the DFT or
            MUSIC');
    end

    %%% Estimate the carrier frequency offset and use to find the rest easier
        %%%
    % Find the max postion, this is most likely within Fs_i/2 Hz of all the
        other frequency offsets
    [~,max_pos] = max(X_f_i);
    % Continuous frequency bins
    F_i = -maxF_i+f_tot(max_pos)*Fs:delF_i:maxF_i+f_tot(max_pos)*Fs;
    % Discrete frequencies
    f_i = F_i./Fs;
    % Recompute weights with less data for faster computations
    if(strcmp(C_spec,'DFT'))
        Wn_i = compute_Wn(f_i, frame_length_i);
    elseif(strcmp(C_spec,'MUSIC'))
        S_i = compute_S(f_i, M_i);
    end

    %%% Use new frequency values to calculate the rest of the Spectrums %%%
    % Empty array to hold spectrogram
    Xlog_i = zeros(length(f_i),n_points_i);
    for i = 1:n_points_i
        % Get the current data frame
        vals = (i-1)*dT_i+1:(i-1)*dT_i+frame_length_i;
        % x is now the frame of dn we are interested in
        x = dn(vals);
        % Apply the proper algorithm to get the frequency spectrum
        if(strcmp(C_spec,'DFT'))
            X_f_i = DFT(x, Wn_i);
        elseif(strcmp(C_spec,'MUSIC'))
            X_f_i = MUSIC(x, P_i, M_i, S_i);
        else
            error('Only supported frequency spectrum algorithms are the DFT or
                MUSIC');
        end
        % Store X_f_i
        Xlog_i(:,i) = 10*log10(abs(X_f_i));
    end

    %%% Now remove the carrier frequency offset %%%
    cpr_path = viterbi(Xlog_i, K_i, F_i, delta_i);
    for i = 1:n_points_i
        % Get the current data frame
        vals = (i-1)*dT_i+1:(i-1)*dT_i+frame_length_i;
        % x is now the frame of dn we are interested in
        x = dn(vals);
        % Modulate x
        x_mod = x.*exp(1j*2*pi*cpr_path(i)*vals'/Fs);
        % Store the modulation in a matrix
        y_mat(:,i) = x_mod;
    end

    %%% Now combine data back together %%%
    % Store new data in variable y
    y = zeros(len, 1);
    for i = 1:n_points_i
        % Get the current data frame
        vals = (i-1)*dT_i+1:(i-1)*dT_i+frame_length_i;
        % Remove mean first to prevent spectral leakage
        y_win = y_mat(:,i) - mean(y_mat(:,i));
        % The plus allows overlap
        y(vals) = y(vals) + y_win;
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% CREATE SPECTROGRAM OF THE DATA %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Create empty spectrograms to hold data
X_f = zeros(length(f),n_points);
X_f_bf = zeros(length(f),n_points);
for i = 1:n_points
    %%% Get the correct data frame depending on the method %%%
    if(Doppler_method == 1)
        % Get the current data frame
        vals = (i-1)*dT+1:(i-1)*dT+frame_length;
        x = d(vals);
    elseif(Doppler_method == 2)
        % Get the current data frame
        vals = (i-1)*dT+1:(i-1)*dT+frame_length;
        x = y(vals);
    else
        error('I do not know how you evaded all my other checks...');
    end
    %%% Transform to frequency domain
    if(strcmp(Spectrum,'DFT'))
        % Compute the DFT
        X_f(:,i) = DFT(x,Wn);
        % Scale up DC term (visual effect only)
        X_f(floor(size(X_f,1)/2)+1,i) = mean(X_f(:,i));
        % Store DFT for the balance factor computation
        X_f_bf(:,i) = X_f(:,i);
    elseif(strcmp(Spectrum,'MUSIC'))
        % Compute MUSIC spectrum
        X_f(:,i) = MUSIC(x,P,M,S);
        % Compute DFT for the balance factor computation
        X_f_bf(:,i) = DFT(x,Wn_cf);
    else
        error('Only supported frequency spectrum algorithms are the DFT or
            MUSIC');
    end
end
X_log = 10*log10(abs(X_f));
```

Listing C.3: Overhead code to perform the path extraction

```
% This file contains the code to extract the Doppler path from a
% Spectrogram
%%%%%%%%%%%%%%%%%%%%%%%%%
%%% FIND BALANCE FACTOR %%%
%%%%%%%%%%%%%%%%%%%%%%%%%
BF = balance_factor(abs(X_f_bf), F);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% APPLY PATH FINDING ALGORITHMS %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Viterbi path %%%
if(strcmp(Path,'viterbi'))
    % Run viterbi algorithm
    best_path = viterbi(X_log, K, F, max_delta_v);
%%% Maximum value path %%%
elseif(strcmp(Path,'max'))
    % Find position of the maximum values
    [~,max_pos] = max(X_log);
    % Path is the frequencies of these maximum values
    best_path = F(max_pos);
%%% Symmetric Viterbi path %%%
elseif(strcmp(Path,'symm_viterbi'))
    % Run the symmetric viterbi algorithm
    symm_path = symmetric_viterbi(X_log, K, F, max_delta_v);
    % Try and find the sign using data from carrier frequency
    % synchronization block
    if(strcmp(Sign_ext,'PLL'))
        best_path = viterbi_sign(symm_path,PLL,ahat,K_sign,dT,floor(sign_DL*Fs
           ));
    % Try and find the sign using data from the symbol synchronization
    % block
    elseif(strcmp(Sign_ext,'SS'))
        best_path = viterbi_sign(symm_path,ss,ahat,K_sign,dT,floor(sign_DL*Fs)
           );
    end
%%% BCJR path %%%
elseif(strcmp(Path,'BCJR'))
    % Run spectrogram through BCJR
    Probs = BCJR(X_log, F, K1, K2, delta_BCJR);
    % Run probability plot through Viterbi
    best_path = viterbi(Probs, K, F, delta_v);
%%% BCJR with balance factor path %%%
elseif(strcmp(Path,'BCJR_BF'))
    % Run spectrogram through BCJR, with the balance factor adjustments
    Probs = BCJR_BF(X_log, F, K1, K2, delta_BCJR, BF, epsilon, K0, sigma, K3,
        Lambda);
    % Run probability plot through Viterbi
    best_path = viterbi(Probs, K, F, delta_BCJR);
%%% No path desired %%%
elseif(strcmp(Path,'none'))
    % Set path to 0 to signify no path
    best_path = 0;
%%% Error check %%%
else
    msg_start = 'Please check that your chosen path is one of the following
        options:';
    msg_end = 'viterbi, max, symm_viterbi, weighted_ave';
    msg = [msg_start newline msg_end];
    error(msg);
end
```

Listing C.4: Overhead code to create spectrogram plots

```
% This file takes care of creating all of the plots from the data output by
% the Doppler extraction system
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PLOT THE TIME DOMAIN SIGNAL %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% First grab the correct data
if(Doppler_method == 1)
        x = d;
elseif(Doppler_method == 2)
        x = dn;
end
% Plot the original signal
if(time_plt ~= 0)
    figure(time_plt); clf();
    % create the time axis values for the whole signal
    t = (0:length(x)-1)/Fs;
    % Plot real portion of the signal
    plot(t,real(x));
    hold on;
    % Plot imaginary portion of the signal
    plot(t,imag(x));
    hold off;
    xlabel('time (s)');
    ylabel('signal magnitude');
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PLOT THE OUTPUT SPECTROGRAM AND BEST PATH %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if((DFT_out_plt ~= 0 && strcmp(Spectrum,'DFT'))||(MUSIC_out_plt ~= 0 && strcmp
    (Spectrum,'MUSIC')))
    if(strcmp(Spectrum,'DFT'))
        spec = figure(DFT_out_plt);
    elseif(strcmp(Spectrum,'MUSIC'))
        spec = figure(MUSIC_out_plt);
    end
    X_log_norm = X_log - max(max(X_log));
    imagesc([-maxF+cenF, maxF+cenF],[0, length(x)/Fs],X_log_norm.');
    colorbar;
    % Plot path
    if(~strcmp(Path,'none'))
        hold on;
        maxT = length(x)/Fs;
        dopp_t = (0:maxT/length(best_path):maxT - maxT/length(best_path));
        plot(best_path, dopp_t, '*r');
        hold off;
    end
    % Plot crest factor decision
    if(show_BF)
        t = 0;
        hold on;
        for bf = BF
            if(abs(bf) > threshold)
                plot(graph_pos+bf*dX,t,'ok','MarkerSize',8);
            else
                plot(graph_pos+bf*dX,t,'xr','MarkerSize',8);
            end
            plot(graph_pos,t,'.b','MarkerSize',8);
            t = t + D_step;
        end
        hold off;
    end
    ylabel('time (s)');
    xlabel('frequency (Hz)')
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% (METHOD2) PLOT THE CARRIER FREQUENCY OFFSET %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot spectrogram of the full spectrum
if(Doppler_method == 2 && carr_offset_plt ~= 0)
    figure(carr_offset_plt); clf();
    imagesc([f_i(1)*Fs, f_i(end)*Fs],[0, length(x)/Fs],Xlog_i.');
    colorbar;
    % Plot the Viterbi path through
    hold on;
    maxT_i = length(x)/Fs;
    dopp_t_i = (0:maxT_i/length(cpr_path):maxT_i - maxT_i/length(cpr_path));
    plot(cpr_path, dopp_t_i, '*r');
    hold off;
    % Graph Aesthetics
    ylabel('time (s)');
    xlabel('frequency (Hz)');
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% BCJR Algorithm Probability output plot %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if((strcmp(Path, 'BCJR')||strcmp(Path, 'BCJR_BF')) && BCJR_plot ~= 0)
    probs = figure(BCJR_plot);
    imagesc([-maxF+cenF, maxF+cenF],[0, length(x)/Fs],10*log10(abs(Probs)).');
    colorbar;
    hold on;
    maxT = length(x)/Fs;
    dopp_t = (0:maxT/length(best_path):maxT - maxT/length(best_path));
    plot(best_path, dopp_t, '*r');
    % Plot crest factor decision
    if(show_BF)
        t = 0;
        hold on;
        for bf = BF
```

```
            if(abs(bf) > threshold)
                plot(graph_pos+bf*dX,t,'ok','MarkerSize',8);
            else
                plot(graph_pos+bf*dX,t,'xr','MarkerSize',8);
            end
            plot(graph_pos,t,'.b','MarkerSize',8);
            t = t + D_step;
        end
        hold off;
    end
    hold off;
    ylabel('time (s)');
    xlabel('frequency (Hz)');
end
```

Listing C.5: Function to get the saved GNU radio output data

```
% GET_COMPLEX_DATA gets complex data from a file
function [data] = get_complex_data(file_name)
    % Open the file
    file = fopen(file_name, 'r');
    % read data into 2 columns
    read = fread(file, [2,Inf], 'float32').';
    % Close the file
    fclose(file);
    % convert data to complex
    data = complex(read(:,1),read(:,2));
end
```

C.2 Spectrum Estimation Functions

Listing C.6: Function to compute the twiddle factors for the DFT

```
%COMPUTE_WN Compute the DFT twiddle factors
%   This function computes the twiddle factors for a certain frequency
%   spectrum to be used on a nfft point DFT
%       f:              The frequencies to compute X_f over (need to be in
%                       discrete form)
%       nfft:           DFT size
%   Outputs:
%       Wn:             The twiddle factors to use
function [Wn] = compute_Wn(f, nfft)
    % Compute bin frequencies from discrete frequencies
    k = f*nfft;
    % Make an empty matrix for Wn
    Wn = zeros(nfft,length(f));

    % Compute the Wn values
    for n = 1:nfft
        Wn(n,:) = exp(-1j*2*pi*k*n/nfft);
    end
end
```

Listing C.7: Function to compute the DFT spectrum

```
%DFT This function computes the DFT algorithm on a signal and returns
%the frequency spectrum
%   This function expects to be given a chunck of data and a set of
%   frequencies that the program will implement the DFT on. It will then
%   return the frequency response
%       input:          A chunck of the input signal to perform the DFT on
%       f:              The frequencies to compute X_f over (need to be in
%                       discrete form)
%       Wn:             The twiddle factors to use
%   Outputs:
%       X_f:            DFT of the signal
function [X_f] = DFT(input, Wn)
    % Check to see that input is a column vector
    if(iscolumn(input))
        % Transpose to a column vector
        input = input';
    end

    % Compute the DFT
    X_f = input*Wn;
end
```

Listing C.8: Function to compute the S vectors for MUSIC

```
%COMPUTE_S Compute the MUSIC S matrix
%   This function computes the S matrid for the music algorithm to speed
%   up the process of computing the algorithm
%       f:              The frequencies to compute X_f over (need to be in
%                       discrete form)
%       M:              M is how big each S vector needs to be
%   Outputs:
%       S:              S matrix for the MUSIC algorithm
function [S] = compute_S(f, M)
    % This is for the length of the y(ti) vectors
    n = 0:M-1;
    % Compute the S matrix
    S = exp(-1j*2*pi*n.'*f);
end
```

Listing C.9: Function to compute the MUSIC spectrum

```
%MUSIC This function computes the MUSIC algorithm on a signal
%   This function expects to be given a chunck of data, from which the
%   program will determine the approximate Ry matrix and use this to apply
%   the MUSIC algorithm to determine the MUSIC spectrum.
%       input:          A chunck of the input signal to perform MUSIC on
%       P:              An expected value for P, the number of expected
%                       signals.
%       M:              A number greater than P.
%       f:              The frequencies to compute P_a over (need to be in
%                       discrete form).
%   Outputs:
%       P_a:            P(f)=1/H(f), the music spectrum of the input
%                       signal.
function [P_a] = MUSIC(input, P, M, S)
    % Check to see that input is a column vector
    if(isrow(input))
        % Transpose to a column vector
        input = input';
    end

    %%% Approximate Ry %%%
    % Find the mean of the signal
    mean_input = mean(input);
    % Subtract mean to get zero-mean noise
    input = input - mean_input;
    % Determine the number of iterations
    N = length(input) - M + 1;
    % Initialize the Ry matrix as an MxM
    Ry_hat = zeros(M);
    for i = 1:N
        %MAC the matricies
        Ry_hat = Ry_hat + input(i:i+M-1)*input(i:i+M-1)';
    end
    % Scale by N
    Ry_hat = Ry_hat./N;

    %%% Now get the eigenvalues and eigenvectors %%%
    [vecs, vals] = eig(Ry_hat);
    % Sort eigenvalues from largest to smallest
    [~, eig_order] = sort(real(diag(vals)), 'descend');
    % Reorder eigenvectors to match reordering of eigenvalues
    eigvecs = vecs(:, eig_order);

    %%% Compute MUSIC spectrum %%%
    size_S = size(S);
    % Empty array to hold frequency respose of MUSIC
    M_a = zeros(size_S(2),1);
    for i = P+1:M
        % MAC the |S(f)'uk|^2 values
        M_a = M_a + abs(S'*eigvecs(:,i)).^2;
    end
    % This create P(f) from M(f)
    P_a = 1./M_a;
end
```

C.3 Viterbi Functions

Listing C.10: Function to compute the Viterbi algorithm for finding the Doppler path

```
%VITERBI This function computes the Viterbi algorithm on a spectrogram
% and returns the optimal path.
%   This function expects a spectrogram input. This function uses this
%   spectrogram to find the optimal path of the Doppler frequency.
%   Inputs:
%       input:      A MxN matrix where M is a collection of frequency bins
%                   and N refers to a time step
%       K:          A weight to determine how much creedance to give to the
%                   branch metric
%       Freqs:      The frequency bins used (in continuous time values)
%       delta:      The maximum frequency difference allowed from current
%                   state to the next state.
%   Output:
%       opt_path    An array containing the optimal path through the data.
function [opt_path] = viterbi(input, K, Freqs, delta)
    % Number of states
    num_states = size(input,1);
    % Number of time steps for Viterbi algorithm
    v_points = size(input,2);
    % Number of paths from one state to the next based on the delta value
    paths_to_state = round(delta/abs(Freqs(1) - Freqs(2)));

    % Empty array for the state paths
    paths = zeros(num_states, v_points);
    % Empty array for the state path values
    M = zeros(num_states,1);

    %%% Compute the initial states %%%
    for m = 1:num_states
        % Start at the given frequency
        paths(m,1) = Freqs(m);
        % Path value initialized to the magnitude of the current frequency
        M(m) = input(m,1);
    end

    %%% Find the most probably path %%%
    % Start at 2 because first point was for intial state
    for j=2:v_points
        % Now loop through each state
        new_M = zeros(num_states, 1);
        temp_paths = zeros(num_states, v_points);
        for m = 1:num_states
            % Get the magnitude at this state
            v = input(m,j);
            % Loop through each path to state m
            for i = -paths_to_state:paths_to_state
                % Check to see if the previous state is within the frequency
                    range
                if((m + i > 0) && (m + i <= num_states))
                    % Compute the branch metric
                    branch_metric = K*abs(Freqs(m+i) - Freqs(m));
                    % Get the previous path value
                    Mprev = M(m+i);
                    % Compute the path value
                    Mnext = Mprev - branch_metric + v;
                    % Compare the path value to the best path value
                    if(Mnext > new_M(m))
                        % If this path value is greater, then record
                        new_M(m) = Mnext;
                        temp_paths(m,1:j) = [paths(m+i,1:j-1), Freqs(m)];
                    end
                end
            end
        end
        % Store the new paths and path values
        M = new_M;
        paths = temp_paths;
    end

    %%% Retreive the state with the best path %%%
    [~, state] = max(M);

    %%% Finally output the best path %%%
    opt_path = paths(state,:);
end
```

Listing C.11: Function to compute the Viterbi algorithm for finding the Doppler path in the presence of symmetry

```
%SYMMETRIC_VITERBI This function computes the Viterbi algorithm on a Matrix of
% frequency data over time and returns the optimal path. This assumes
% symmetry in the negative and positive frequencies.
%   This function expects an input of a spectrogram. This function uses
%   this data to find the optimal path a Doppler frequency. This function
%   assumes symmetry in the positive and negative frequencies, and it only
%   tracks the positve frequency, using the symmetry as additional
%   information to help track.
%   Inputs:
%       input:      A MxN matrix where M is a collection of frequency bins
%                   and N refers to a time step
%       K:          A weight to determine how much credence to give to the
%                   branch metric
%       Freqs:      The frequency bins used (in continuous time values)
%       delta:      The maximum frequency difference allowed from current
%                   state to the next state.
%   Output:
%       opt_path    An array containing the optimal path through the data.
function [opt_path] = symmetric_viterbi(input, K, Freqs, delta)
    % Number of states, assuming 0 is the center frequency and there are
    % the same number of frequencies on each side
    num_states = floor(size(input,1)/2)+1;
    % Number of time steps for Viterbi algorithm
    v_points = size(input,2);
    % Number of paths from one state to the next based on the delta value
    paths_to_state = round(delta/abs(Freqs(1) - Freqs(2)));

    % Empty array for the state paths
    paths = zeros(num_states, v_points);
    % Empty array for the state path values
    M = zeros(num_states,1);

```

```
%%% Compute the initial states %%%
for m = 1:num_states
    % Start at 0 (num_states) and then add m-1 to it to get all positive
        values
    paths(m,1) = Freqs(num_states+m-1);
    % Initial path value adds both the positive and negative path values
    M(m) = input(num_states+m-1,1)+input(num_states-(m-1),1);
end

%%% Find the most probably path %%%
% Start at 2 because first point was for intial state
for j=2:v_points
   % Now loop through each state
   new_M = zeros(num_states, 1);
   temp_paths = zeros(num_states, v_points);
   for m = 1:num_states
       % Loop through each path to state m
       for i = -paths_to_state:paths_to_state
           % Check to see if the previous state is within the frequency
               range
           if((m + i > 0) && (m + i <= num_states))
               % Compute the branch metric
               branch_metric = K*abs(Freqs(num_states+m-1+i) - Freqs(
                   num_states+m-1));
               % Get the previous path value
               Mprev = M(m+i);
               % Compute V(f) = magnitude at both (+) and (-) f
               V_f = input(num_states+m-1,j) + input(num_states-(m-1),j);
               % Compute the path value
               Mnext = Mprev - branch_metric + V_f;
               % Compare the path value to the best path value
               if(Mnext > new_M(m))
                   % If this path value is greater, then record
                   new_M(m) = Mnext;
```

```
                        temp_paths(m,1:j) = [paths(m+i,1:j-1), Freqs(num_states
                            +m-1)];
                    end
                end
            end
        end
        % Store the new paths and path values
        M = new_M;
        paths = temp_paths;
    end

    %%% Retreive the state with the best path %%%
    [~, state] = max(M);

    %%% Finally output the best path %%%
    opt_path = paths(state,:);
end
```

Listing C.12: Function to compute the Viterbi algorithm for finding the sign of the Doppler

```
%VITERBI_SIGN implements a viterbi algorithm to find the sign of the
% Doppler frequencies
%   This function takes in a collection of Doppler frequencies and the
%   symbol synchronizer output to determine the sign of the Doppler
%   frequencies.
%   Inputs:
%       dopp_frequencies:   A list of the absolute values of the Doppler
%                           frequencies
%       sk:                 The output data of the symbol synchronizer
%       ak:                 The output data of the symbol decision block
%       K:                  Constant to adjust how much weight to give the
%                           branch metric
%       dT:                 Step size from data point to data point
%       Data_length:        Length of data that goes into each Doppler
%                           frequency.
%   Outputs:
%       best_path:          The Doppler frequency path
function [best_path] = viterbi_sign(dopp_frequencies,sk,ak,K,dT,Data_length)
    % Number of states, one for (+) and one for (-)
    num_states = 2;
    % Number of time steps for Viterbi algorithm
    v_points = size(dopp_frequencies,2);

    % Empty array for the state paths
    paths = zeros(num_states, v_points);
    % Empty array for the state path values
    M = zeros(num_states,1);

    %%% Find the most probably path %%%
    for i = 1:v_points
        % Get the indecies for the data points that goes into this frequency
        val = (i-1)*dT+1:(i-1)*dT+Data_length;
        % Grab the data from the symbol synchronizer needed for this Doppler
            frequency
        si = sk(val);
        % Grab the data from the symbol decision needed for this Doppler
            frequency
        ai = ak(val);

        %%% Calculate values for the radius and theta %%%
        % Average radius
        r = sum(abs(si))/length(si);
        % The radius of the doppler rotation
        rad = abs(si) - r;
        % The angle of the doppler rotation
        theta = unwrap(angle(si) - angle(ai));

        %%% Use a linear line fit to theta to remove the linear portion of the
             data %%%
        % Create a A vector, 2 x N where the first column is the x values and
            the second all ones
        A = [val.', ones(length(val),1)];
        % This computes (((A^T)A)^-1)(A^T)y, which solves for m, the slope,
            and b, the offset, of the line
        c = A\theta;
        % Remove the linear fit to get only the Doppler and noise on top
        theta = theta - (val.'.*c(1) + c(2));

        %%% Load in an advance filter and use to advance theta and rad %%%
        % (filter designed using the matlab filter design tool and stored)
        load('advance_filter.mat');
        adv_theta = filter(adv_filt{1}, adv_filt{2}, theta);
        adv_rad = filter(adv_filt{1}, adv_filt{2}, rad);

        %%% Correlate the advanced signals to determine the probable sign %%%
        advan = corr(adv_theta, rad);
        delay = corr(adv_rad, theta);

        %%% Viterbi to determine most likely sign
        if(i==1)    % This is for the intial state
            % The initial state is the result of the first correlation
            M(1) = delay;
            M(2) = advan;
            % The state output is +/- the first frequency
            paths(1,1) = -dopp_frequencies(1);
            paths(2,1) = dopp_frequencies(1);
        else        % All other states times
            % Initialize temporary matricies to hold the updated
            % information
            new_M = zeros(num_states,1);
            temp_paths = zeros(num_states, v_points);
            for j = 1:num_states
                % Compute the branch metrics
                neg_branch_metric = K*abs(-dopp_frequencies(i-1) - (j*2-3)*
                    dopp_frequencies(i));
                pos_branch_metric = K*abs(dopp_frequencies(i-1) - (j*2-3)*
                    dopp_frequencies(i));
                % Get the previous path values
                neg_prev = M(1);
                pos_prev = M(2);
                % Compute the path values
                neg_Mnext = pos_prev - neg_branch_metric + delay;
                pos_Mnext = pos_prev - pos_branch_metric + advan;
                % Determine which path is more likely by the path value
                if(pos_Mnext > neg_Mnext)
                    % The new state is most likely (+)
                    new_M(j) = pos_Mnext;
                    temp_paths(j,1:i) = [paths(2,1:i-1), dopp_frequencies(i)];
                else
                    % The new state is most likely (-)
                    new_M(j) = neg_Mnext;
                    temp_paths(j,1:i) = [paths(1,1:i-1), -dopp_frequencies(i)
                        ];
                end
            end
            % Replace path value and path with the temporary variables
            M = new_M;
            paths = temp_paths;
        end
    end
    % Retreive the state with the best path
    [~, state] = max(M);

    % Return the best path calculation
    best_path = paths(state,:);
end
```

C.4 BCJR Functions

Listing C.13: Function to compute the BCJR algorithm

```
%BCJR creates a probabilty matrix from a spectrogram of data
%   This function takes in a spectrogram of data and calculates the
%   probabilty of the Doppler at each frequency overtime.
%   Inputs:
%       input:      Frequency data over time
%       Freqs:      The frequency states at each time period
%       K1:         Constant to give weight to the branch metric
%       K2:         Constant to give weight to the state value
%       delta:      The maximum distance for a frequency change
%                   from time period to time period
%   Output:
%       P:          Matrix of the probabilities of each state at each time
%                   step
function [P] = BCJR(input,Freqs,K1,K2,delta)
    % Total number of states
    num_states = size(input,1);
    % The number of time steps
    v_points = size(input,2);

    %%% Pre-calculate part of the gammas, the branch metric portion %%%
    % Create an empty array
    G = zeros(num_states,num_states);
    for i = 1:num_states
        for j = 1:num_states
            % This is the frequency shift from state i to state j
            del = abs(Freqs(i) - Freqs(j));
            % If within the maximum state shift values allowed
            if(del <= delta)
                % Set the G value at this state difference to  e^(-K|f1-f2|)
                G(i,j) = exp(-K1*del);
            % If not within the maximum states shift values allowed
            else
```

```
                % Set G to 0
                G(i,j) = 0;
            end
        end
    end

    %%% Initialize alpha and betas %%%
    % Matrix to store the alpha values over time
    A = zeros(num_states,v_points);
    % Set all of the initial alphas to 1/Q
    A(:,1) = 1/num_states;
    % Matrix to store the beta values over time
    B = zeros(num_states,v_points);
    % Set all the initial betas to 1/Q
    B(:,end) = 1/num_states;

    %%% Run forward and backward passes at the same time %%%
    for t = 1:v_points-1
        %%% Forward pass %%%
        % Column vector of data at time t
        forward_data = input(:,t);
        % Column vector of current alphas
        alpha = A(:,t);
        % QxQ matrix of gamma at time t, p indexed along row q along column
        gamma = G.*exp(K2*forward_data).';
        % Next alpha is a sum of all the paths from p to q
        alpha_tilde = gamma*alpha;
        % Normalize
        A(:,t+1) = (1/sum(alpha_tilde))*alpha_tilde;
        %%% Backward pass %%%
        % Column vector of data at time N-t
        backward_data = input(:,v_points-t);
        % Column vector of current betas
        beta = B(:,v_points-t+1);
        % QxQ matrix of gamma at time N-t, p indexed along row q along column
        gamma = G.*exp(K2*backward_data.');
        % Next beta is a sum of all the paths from p to q
        beta_tilde = gamma*beta;
        % Normalize
        B(:,v_points-t) = (1/sum(beta_tilde))*beta_tilde;
    end

    %%% Now calculate the path probabilities %%%
    % Probability of the state at this time
    P = zeros(num_states,v_points);
    for t = 1:v_points-1
        % Column vector of data at time t
        data = input(:,t);
        % Column vector of alpha at time t
        alpha = A(:,t);
        % Column vector of beta at time t+1
        beta = B(:,t+1);
        % Matrix of gamma using current data
        gamma = G.*exp(K2*data);
        % Probabilities of each state at this time
        P(:,t) = alpha.*(gamma*beta);
    end
end
```

Listing C.14: Function to compute the BCJR algorithm with balance factor adjustment

```
%BCJR_BF creates a probabilty matrix from a spectrogram of data
%   This function takes in a spectrogram of data and calculates the
%   probabilty of the Doppler at each frequency overtime, taking into
%   account the balance factor.
%   Inputs:
%       input:      Frequency data over time
%       Freqs:      The frequency states at each time period
%       K1:         Constant to give weight to the branch metric
%       K2:         Constant to give weight to the state value
%       delta:      The maximum distance for a frequency change
%                   from time period to time period
%       BF:         A vector containing the balance factor for each time
%                   step
%       epsilon:    The threshold variable for determining whether or not
%                   to pay attention to the balance factor
%       K0:         Constant to give weight to the balance factor
%       sigma:      Width of the adjustment spectrum
%       K3:         Controls how fast the magnitude of the adjustment
%                   spectrum linearly drops away from the maximum at 0 Hz
%       Lambda:     The maximum height of the adjustment spectrum at 0 Hz
%   Output:
%       P:          Matrix of the probabilities of each state at each time
%                   step
function [P] = BCJR_BF(input,Freqs,K1,K2,delta,BF,epsilon,K0,sigma,K3,Lambda)
    % Total number of states
    num_states = size(input,1);
    % The number of time steps
    v_points = size(input,2);

    %%% Create the adjustment spectrum %%%
    V0 = zeros(num_states,1);
    % Cycle through all values you want to give weight to
    for i = -sigma:sigma
        V0(floor(num_states/2)+1+i) = Lambda-abs(i*K3);
    end

    %%% Pre-calculate part of the gammas, the branch metric portion %%%
    % Create an empty array
    G = zeros(num_states,num_states);
    for i = 1:num_states
        for j = 1:num_states
            % This is the frequency shift from state i to state j
            del = abs(Freqs(i) - Freqs(j));
            % If within the maximum state shift values allowed
            if(del <= delta)
                % Set the G value at this state difference to  e^(-K|f1-f2|)
                G(i,j) = exp(-K1*del);
            % If not within the maximum states shift values allowed
            else
                % Set G to 0
                G(i,j) = 0;
            end
        end
    end

    %%% Initialize alpha and betas %%%
    % Matrix to store the alpha values over time
    A = zeros(num_states,v_points);
    % Set all of the initial alphas to 1/Q
    A(:,1) = 1/num_states;
    % Matrix to store the beta values over time
    B = zeros(num_states,v_points);
    % Set all the initial betas to 1/Q
    B(:,end) = 1/num_states;

    %%% Run forward and backward passes at the same time %%%
    for t = 1:v_points-1
        %%% Forward pass %%%
        if(abs(BF(t)) > epsilon)
```

```
            % Column vector of data at time t
            forward_data = input(:,t);
        else
            % Column vector of data at time t taking into account the
            % balance factor
            BF_tilde = abs(BF(t))*K0;
            forward_data = BF_tilde*input(:,t) + (1-BF_tilde)*V0;
        end
        % Column vector of current alphas
        alpha = A(:,t);
        % QxQ matrix of gamma at time t, p indexed along row q along column
        gamma = G.*exp(K2*forward_data).';
        % Next alpha is a sum of all the paths from p to q
        alpha_tilde = gamma*alpha;
        % Normalize
        A(:,t+1) = (1/sum(alpha_tilde))*alpha_tilde;
        %%% Backward pass %%%
        if(abs(BF(t)) > epsilon)
            % Column vector of data at time N-t
            backward_data = input(:,v_points-t);
        else
            % Column vector of data at time N-t taking into account the
            % balance factor
            BF_tilde = abs(BF(t))*K0;
            backward_data = BF_tilde*input(:,v_points-t) + (1-BF_tilde)*V0;
        end
        % Column vector of current betas
        beta = B(:,v_points-t+1);
        % QxQ matrix of gamma at time N-t, p indexed along row q along column
        gamma = G.*exp(K2*backward_data.');
        % Next beta is a sum of all the paths from p to q
        beta_tilde = gamma*beta;
        % Normalize
        B(:,v_points-t) = (1/sum(beta_tilde))*beta_tilde;
    end

    %%% Now calculate the path probabilities %%%
    % Probability of the state at this time
    P = zeros(num_states,v_points);
    for t = 1:v_points-1
        if(abs(BF(t)) > epsilon)
            % Column vector of data at time t
            data = input(:,t);
        else
            % Column vector of data at time t taking into account the
            % balance factor
            BF_tilde = abs(BF(t))*K0;
            data = BF_tilde*input(:,t) + (1-BF_tilde)*V0;
        end
        % Column vector of alpha at time t
        alpha = A(:,t);
        % Column vector of beta at time t+1
        beta = B(:,t+1);
        % Matrix of gamma using current data
        gamma = G.*exp(K2*data);
        % Probabilities of each state at this time
        P(:,t) = alpha.*(gamma*beta);
    end
end
```

Listing C.15: Function to compute the balance factors

```
%BALANCE_FACTOR Computes a spectrum balance factor
%   This takes in a frequency spectrum and finds the energy balance between
%   the positive and negative frequencies.
function [BF] = balance_factor(freq_spectrum, freqs)
    % Find where the the DC term is
    zero_pos = find(freqs==0);

    % Find the sum of each side and the total spectrum
    L_sum = sum(freq_spectrum(1:zero_pos, :), 1);
    R_sum = sum(freq_spectrum(zero_pos:end, :), 1);
    T_sum = sum(freq_spectrum, 1);

    % Figure out how much energy is assigned to each side
    fl = L_sum./T_sum;
    fr = R_sum./T_sum;

    % Subtract the energy balances to get a balance factor
    % BF ranges from -1 to 1, -1 meaning all negative, 1 meaning all
    % positive, -1 meaning all negative, 0 meaning balanced
    BF = fr - fl;
end
```