Cooper Union

Albert Nerken School of Engineering

Synchronization of Interference to Facilitate Joint Detection

by

Andrew Apollonsky

A thesis submitted in partial fulfillment

of the requirements for the degree of

Master of Engineering

May 5, 2014

Doctor Sam Keene, Advisor

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

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

_____

_____

Acknowledgements

I would like to thank my advisor, Dr. Sam Keene, for his guidance and general availability throughout this process. Without him pushing me, I'm certain that I would not be where I am today. I would like to thank the rest of the faculty, friends and peers at the Cooper Union Albert Nerken School of Engineering. A special thanks is due to Nicolas Avrutin, who listened to some of my ideas and offered some of his own. Lastly, I would like to thank my family; I would certainly be nowhere near here without my parents, Alex and Nataly Apollonsky, my brothers Dimitry and Daniel, and my sister Emilie.

Abstract

Interference-blind demodulation techniques fail often and lead to performance degradation in systems with routine frame collisions. Performing joint detection on two interfering frames, which would allow for the decoding of both despite the collision, could yield performance improvements in wireless communication systems. This would lead to fewer retransmissions, conserving system power, and allow for the replacement of current MAC-layer collision management mechanisms such as the 802.11 RTS/CTS, which introduce performance degredation due to overhead.

This paper proposes a system capable of resolving a frame collision with a time-domain QAM modulation scheme in a quasi-static flat-fading channel with Additive White Gaussian Noise (AWGN). The system detects the presence of the interferer in the frame currently being decoded, and performs frequency, phase and gain synchronization on the interferer while continuing to decode the primary frame. Once both frames are synchronized, a joint detection algorithm is used to complete decoding of both frames. Performance of the system is analyzed, and potential avenues for improvement in future work are explored.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation for Research

The growth of wireless systems is expected to continue at a breakneck pace for the next decade. Research efforts into expanding the capabilities of modern systems like the 802.11 standard have focused on improving the efficacy of spatial multiplexing using multiple-input and multiple-output (MIMO) systems [3] or extending the technology into higher frequency bands [4]. However, many standards — particularly those for random access networks — suffer from data loss due to frame collisions. Collisions necessitate retransmission of the colliding frames, expending additional time and power. One such cause for collisions is the hidden node problem, wherein two transmitters may both be in range of the receiver but not in range of each other. When this occurs in the 802.11 standard and one frame is at a higher power than the other, then the weaker frame is dropped, as it has a negligible effect on the received signal; however, if they are both of

a similar power, both frames are rendered undecodable due to the interference [12].

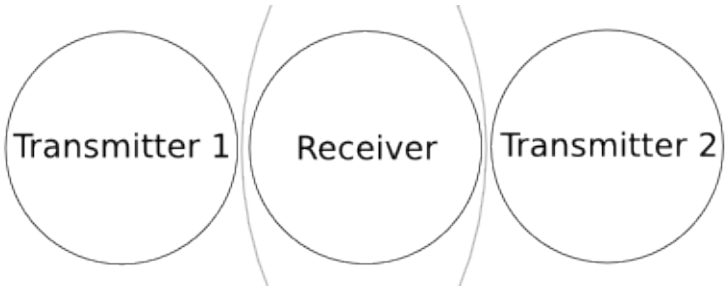

Figure 1.1: Frame Collision Condition

The 802.11 standard attempts to address this problem by including the request-to-send/clear-to-send (RTS/CTS) mechanism, in which if the data packet is sufficiently long, transmitters send a short request-to-send (RTS) frame to the receiver to acquire permission to begin transmission of a larger frame. The receiver times the acknowledging clear-to-send (CTS) packets to ensure that when the full frames are transmitted, they would not interfere with each other. There are several problems with this approach. Firstly, the RTS/CTS mechanism causes overhead, degrading system performance even when network congestion is low and collisions unlikely. Furthermore, the range over which the receiver can return a CTS is smaller than that over which interference could make a transmitted frame unrecoverable [2], making it an imperfect solution even in congested networks. Alternative means of resolving collisions would then allow for higher throughput in 802.11 networks, and a different receiver architecture is required to facilitate the detection of two interfering frames. Research to solve the problem of interference is being done along several lines — including decoder architectures like ZigZag [7], collaborative transmission like Interference Alignment [9], and joint decoding of multiple transmissions [1].

1.1.1 ZigZag

One potential solution to this problem aimed squarely at the 802.11 case is ZigZag, a receiver that can use the information received by receiving two separate collisions between two packets to decode both. The algorithm requires that there are two separate collisions of the same packets, which makes it a solution to only a small subset of all collisions. One of the strengths of the algorithm is that it requires no changes to the 802.11 MAC and introduces no overhead in the case of no collision. When packets collide for a second time, ZigZag achieves the same performance as if the packets were scheduled in separate time slots.

ZigZag exploits a basic characteristic of the 802.11 standard to achieve decoding. 802.11 transmitters wait a random period of time when retransmitting a frame that was lost, meaning that the chances of two frames colliding a second time with the same bits of each frame interfering are very small. This also implies that several of the symbols of the frame will arrive without interference.

Over the course of two collisions with random time offsets, then, the receiver will have sufficient information to decode "chunks" of these frames. These chunks can then be used to decode different chunks in the other collision. The functionality of the ZigZag algorithm is shown in Fig. 1.2.

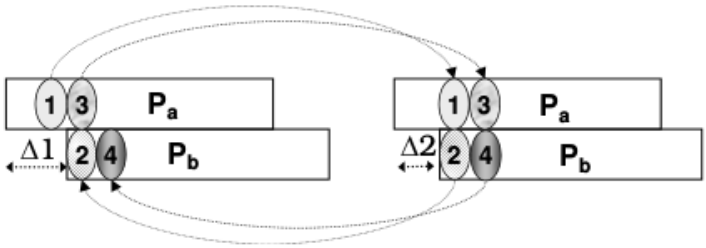

Figure 1.2: ZigZag first decodes chunk one in the first collision, uses it in the second collision to decode chunk two, which it uses to decode chunk 3 in the first collision, etc. [7]

The ZigZag algorithm is simple and elegant, but as noted previously, is only useful when a collision has deteriorated to such a state that the same two frames collide twice. The algorithm cannot be applied to decode single collisions. This limits its usefulness; though it can potentially be applied, it does not completely solve the problem.

1.1.2 Interference Alignment

Other approaches to interference mitigation focus on overhauling the entire system, including novel transmission schemes. One such technique is interference alignment, in which multiple transmitters coordinate their transmissions such that the mutual interference aligns at the receivers [9]. The possibilities of this approach are demonstrated in [11], where the authors show that interference alignment leads to a linear increase in total system capacity with the number of users.

Interference alignment is mathematically very promising, but in practice, there are many obstacles that preclude its implementation. Probably the largest obstacle is one of synchronization. In order for a transmitter to precode its transmission in such a way that all undesired receivers can discard it, the transmitter must have very accurate channel

state information (CSI) about the link with every receiver. Unlike the typical transmission, where synchronization is fully handled on the receiver side and so no feedback is necessary, there must some feedback to the transmitter so that it could estimate the CSI for itself. These additional training symbols complicate the system architecture and lower the maximum system throughput. Similarly, in order for multiple transmitters to utilize the available capacity to its fullest extent, the transmitters must communicate *with each other* to ensure that they are utilizing all available degrees of freedom in the system.

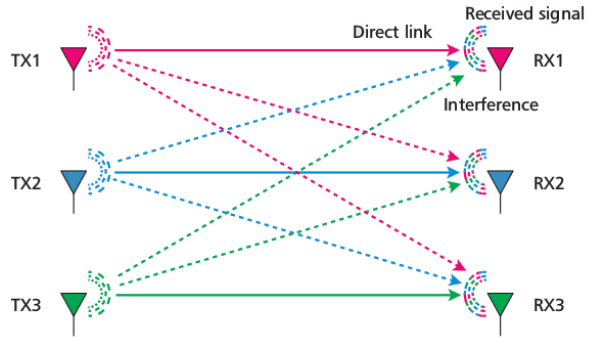

Figure 1.3: Interference Alignment. The transmitters construct their waveforms such that they are clearly seen as interference by unintended recipients. [10]

Interference alignment is a promising technology that could potentially be implemented in the future, but it is not be compatible with modern standards like 802.11. Unlike ZigZag, it requires modification of the transmitter for implementation.

1.1.3 Joint Detection

A potential solution to the hidden node problem is joint detection, explained in [1]. The authors examine several methods to decode two interfering streams. One of them

was successive interference cancellation, or SIC. With SIC, the receiver would first treat one of the two symbols as noise to demodulate the other, and then use that symbol to estimate the first symbol. The authors also developed an ideal Joint Maximum-Likelihood (JML) detector that could decode one of the symbols by using knowledge of the constellation of the other, but the JML detector was deemed too computationally expensive for practical implementation. To this end, the authors developed a simplified Joint Minimum-Distance (JMD) detector, which yielded near-ML performance despite being far less computationally expensive. In essence, the detector looked through every possible combination of symbols sent by the two transmitters for every sample, and picked the one that was closest to the received symbol.

The authors compared the different detection schemes to the interference-ignorant demodulation scheme, and concluded that there were very significant performance improvements. JML proved to be the most accurate, with JMD providing near-ML performance and successive interference cancellation trailing far behind. The authors showed that unlike for the interference-ignorant and SIC approaches, there was no BER error floor for the JML and JMD detectors, and so as the SNR of the signals increased, the BER would continue to improve. In essence, by using the available information about the constellation of the interferer, the detectors made the detection rates limited not by the SINR, but purely by the SNR.

However, the authors made the assumption of perfect synchronization. They assumed that the receiver has a priori knowledge about the channel gains, the frequency offsets, and that there was no offset in time between the two frames, such that the interfer-

ing symbols lined up perfectly. These assumptions are not valid for a real communication system, and are addressed in this thesis. This deficiency in the work inspired the system developed here.

The main result of this work is a system that detects the beginning of an interfering frame during the decoding of a primary frame, and performs synchronization and decoding on the interfering frame while continuing to decode the primary frame. The work assumes a quasi-static fading environment with AWGN. An 802.11-like frame structure is assumed, where each frame includes twelve training symbols which form the basis of all synchronization.

Chapter 2

Communication Basics

Digital data are typically expressed as a stream binary bits, represented by 1s and 0s. The purpose of a communication system is to transmit these binary bits to the designated receiver, which can be done by wire, which is more reliable but necessitates a physical link, or wirelessly, which is more complicated but more flexible after implementation. Our focus will be on wireless communications, though many of the principles can also be applied to wired communications.

In order to transmit the data over a wireless channel, the data must somehow be converted into a form that can be transmitted and received — it must be converted into an electromagnetic wave. Typical transmitter hardware generates a generic waveform, which is then modulated, which allows for the "imprinting" of the data onto the wave by modifying its properties. The receiver reverses the imprinting process, extracting the data from the waveform. We will briefly cover the basics of modulation here, setting the foundation for the remainder of this thesis.

2.1 Digital Modulation

Digital modulation entails modulating an analog carrier signal with a discrete signal (in our case, the data to be transmitted). There are several ways to do this, some of which will be covered here.

2.1.1 Pulse Amplitude Modulation (PAM)

Digital PAM, as the name suggests, encodes the information in the amplitude of the waveform being transmitted. This can be expressed as

$$s_m(t) = A_m p(t), 1 \leq m \leq M \tag{2.1}$$

where m specifies which of M possible amplitudes can be transmitted. A_m refers to the amplitude itself, and $p(t)$ represents the pulse of duration T. In a simple PAM system with only two possible amplitudes (and $log_2(2) = 1$ bits/symbol), a 0 may correspond to an amplitude of –1 and a 1 to an amplitude of 1.

This is an accurate description of the system at baseband, but this is not the pulse that is wirelessly transmitted. The modulated pulse is upconverted to a higher frequency prior to transmission. In this generic form of PAM signalling, $p(t)$ can be replaced with $g(t)cos(2\pi f_c t)$, where f_c is the carrier frequency and $g(t)$ the baseband pulse waveform.

In this case,

$$s_m(t) = Re[s_{ml}(t)e^{j2\pi f_c t}] \tag{2.2}$$

$$= Re[A_m g(t)e^{j2\pi f_c t}] \tag{2.3}$$

$$= A_m g(t)cos(2\pi f_c t) \tag{2.4}$$

An example of a PAM waveform is given in Fig. 2.1.

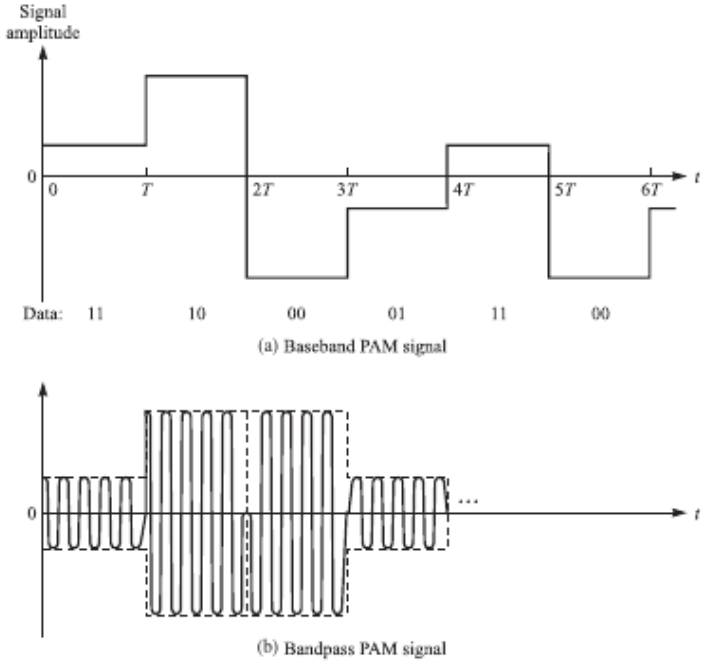

Figure 2.1: Example of (a) baseband and (b) carrier-modulated PAM signals [15, Fig. 3.2-2]

2.1.2 Phase-Shift Keying (PSK)

Phase-shift keying is the discrete form of pure phase modulation. In phase-shift keying waveforms, thus, the amplitude and frequency of the transmitted wave remain constant, and data is transmitted by changing the phase of the transmitted waveform.

The M signal waveforms corresponding to the M different potential symbols transmitted can be represented as

$$s_m(t) = Re[g(t)e^{\frac{2\pi(m-1)}{M}}e^{j2\pi f_c t}], \quad m = 1, 2, ..., M \tag{2.5}$$

$$= g(t)cos(2\pi f_c t + \frac{2\pi}{M}(m-1)] \tag{2.6}$$

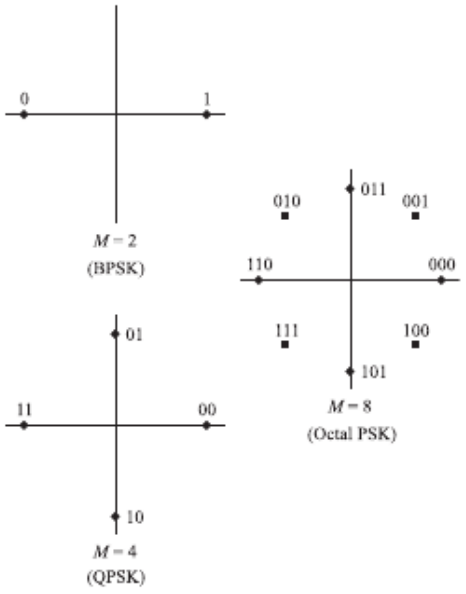

Figure 2.2: Constellation diagrams for M=2, M=4 and M=8. [15, Fig. 3.2-3]

In theory, it is possible to increase the potential number of symbols — and, correspondingly, the number of bits per symbol — arbitrarily high. However, the downside to doing so is that the transmitted symbols become more vulnerable to the effects of channel distortion and AWGN. Trying to transmit symbols from a constellation in which there is a 2^o difference between symbols is a fruitless affair when the channel randomly shifts the received symbol by upwards of 2^o 50% of the time. A useful metric for gauging how resilient a constellation is to AWGN of a certain power is the d_{min}, which represents the minimal distance in the signal space diagrams between two symbols. Signal space diagrams of several PSK constellations are shown in 2.2.

As seen, as M increases, d_{min}, the minimal distance between two transmitted symbols, decreases, and so the constellation becomes less resilient to noise. Thus, the prob-

ability of a symbol error occuring increases. Note that an increase in the symbol power would effectively push all the points of the PSK constellation outwards from the center, increasing d_{min} and reducing the probability of an error at the expense of higher power requirements.

2.1.3 Quadrature Amplitude Modulation (QAM)

We have observed that information can be transmitted in both phase and amplitude. It makes sense, then, that information can also be transmitted by modulating both simultaneously, which Quadrature Amplitude Modulation, or QAM, does. A QAM signal can be expressed either in a polar or in a cartesian form. The cartesian form is given by

$$s_m(t) = Re[(A_{mi} + jA_{mq})g(t)e^{j2\pi f_c t}] \tag{2.7}$$

$$= A_{mi}g(t)cos(2\pi f_c t) - A_{mq}g(t)sin(2\pi f_c t], \quad m = 1, 2, ..., M \tag{2.8}$$

where A_{mi} and A_{mq} are signal amplitudes of the quadrature carriers and $g(t)$ is the signal pulse. The equivalent polar form is

$$s_m(t) = Re[r_m e^{j\theta_m} e^{j2\pi f_c t}] \tag{2.9}$$

$$= r_m cos(2\pi f_c t + \theta_m) \tag{2.10}$$

where $r_m = \sqrt{A_{mi}^2 + A_{mq}^2}$ and $\theta_m = tan^{-1}(\frac{A_{mq}}{A_{mi}})$. This expression makes it clear that QAM modulation can be viewed as a combination of phase and amplitude modulation.

In theory, there are no limits on what points on a signal space diagram could be

combined to form a viable constellation for modulation. However, power, symbol error and transmission rate constraints favor constellations with the largest d_{min} for the least power. The most common constellation diagrams for QAM are simple rectangular grids of potential symbols, each equidistant from adjacent symbols.

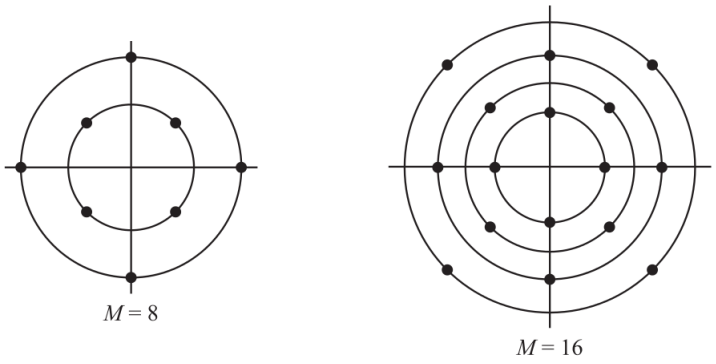

Figure 2.3: Constellation diagram of QAM modulator. [15, Fig. 3.2-4]

Fig. 2.3 shows that there is room for variation, though there is typically little incentive for abnormal constellation due to potential issues during implementation in hardware and very limited performance benefits.

2.2 Signal Distortion

The waveform that a receiver in a wireless digital communication system attempts to decode can differ greatly from the waveform that was sent by the transmitter, or even from the waveform that the transmitter was designed to send. Synchronization is the process of the receiver undoing distortions introduced by the channel, paving the way for demodulation of the waveform to extract the transmitted data.

In a typical communication system, this is achieved by the transmission of training

symbols at the beginning of a frame. The training symbols are known in advance by the receiver, being specified in the communication standard. The receiver observes that a signal is present, and uses its knowledge of what the signal should be transmitting to estimate and undo the effects of the channel. This section will briefly cover the effects of the channel on the transmitted signal. To recover the original data — learn what was done by the channel, and invert it!

2.2.1 Path Loss

Consider a generic QAM signal described by

$$s_m(t) = r_m cos(2\pi f_c t + \theta_m) \tag{2.11}$$

If the transmitter has an ideal non-directional antenna, then the waveform is sent in all directions. The wavefront can then be viewed as a sphere, with the power being dispersed equally across the surface area of the sphere. As the surface area of the sphere scales linearly with the square of the distance from the center, we can then conclude that the power of the transmitted wave decreases with the square of distance, and so the amplitude of the signal at the receiver is reduced. For long-distance wireless communications, such as a satellite communications system, this attenuation can exceed 100dB, necessitating high transmission power or low data rates to ensure that the data could be recovered at the receiver. The losses due to propogation can be reduced by using a directional antenna, but this is a rare approach in certain wireless systems (like Wifi) due to the difficulty in targeting the transmission towards the receiver, when either of the two could

potentially be moving.

2.2.2 Shadowing

In most communication systems, a direct line of sight is not guaranteed, and the signal may be further attenuated by blockage from objects in the signal path. The attenuation caused by these objects, or fading, may vary with time, position or frequency, and is thus often modeled as a random process.

The effects of shadowing greatly depend on the materials comprising the blockage of line of sight; there is virtually no additional loss due to a signal passing through air, whereas a stone wall can greatly attenuate the waveform. The attenuation due to fading also tends to increase linearly with frequency.

The log-normal shadowing model is often used to model the effects of shadowing in both indoor and outdoor environments [14, pp. 48]. In the log-normal shadowing model, the ratio of transmit to receiver power $\psi = \frac{P_t}{P_r}$ is assumed random, and is given by

$$p(\psi) = \frac{\frac{10}{ln10}}{\sqrt{2\pi}\sigma_{\psi dB}\psi} exp\left(-\frac{(10log_{10}\psi - \mu_{\psi dB})^2}{2\sigma^2_{\psi dB}}\right) \tag{2.12}$$

where $\mu_{\psi dB}$ is the mean and $\sigma^2_{\psi dB}$ is the variance of $\psi_{dB} = 10log_{10}\psi$. $\mu_{\psi dB}$ depends primarily on the path loss, whereas $\sigma^2_{\psi dB}$ depends on the variation in path loss — typically caused by movement in the environment.

2.2.3 Phase Offset

Consider again the signal in Eqn. 2.11. The waveform takes a finite amount of time to arrive at the receiver, and as seen from the signal equation, the value that would be "seen" is time-varying. As such, the constellation seen at the receiver will be rotated by some angle. Both the phase offset and the effects of flat fading are typically modeled as a multiplication of the transmitted symbol by a complex constant.

2.2.4 Doppler Spread

Frequency offsets can occur along the transmission chain. They may be caused by an offset in the oscillators in either the transmitter or the receiver, or they may be caused by other factors such as movement of the transmitter or the receiver. These frequency offsets are referred to as doppler shifts. The difference in doppler shifts between different components of the signal contributing to a single channel is referred to as a Doppler Spread. The doppler spread is inversely proportional to the coherence time of the channel; as the doppler spread increases, the typical frequency offset increases, which leads to more rapid changes in effects of the channel and a decrease in the coherence time.

2.2.5 Multipath

Multipath refers to the possibility that of the original waveform, several copies arrive at the receiver at different times. This may occur due to reflections of the electromagnetic waves off of walls, trees, buildings, the ground, or anything else. These copies of the signal interfere with other copies, making correct decoding of the transmitted signal

more difficult.

The multipath channel can also be (and typically is) time-dependent; this is exacerbated by movements of the transmitter or receiver, which lead to a change of the path lengths and reflections. Because time variations appear unpredictable to the user, it is reasonable to statistically characterize multipath channels.

Consider a generic electromagnetic wave:

$$s(t) = Re[s_l(t)e^{j2\pi f_c t}] \tag{2.13}$$

In the event that there are multiple propagation paths, the receiver sees the sum of the signals along all paths, each of which has been attenuated and delayed by some amount. The received bandpass signal can be expressed as

$$x(t) = \sum_n \alpha_n(t)s[t - \tau_n(t)] \tag{2.14}$$

$\alpha_n(t)$ is here the attenuation factor and $\tau_n(t)$ is the propagation delay associated with the nth path. An equivalent highpass signal with appropriate accounting for phase can be expressed by multiplying the $x(t)$ by $e^{j2\pi f_c t}$ and taking the real part. To account for the phase offset caused by the additional propagation delay, each path must also be multiplied by $e^{-j2\pi f_c \tau_n(t)}$; thus,

$$x(t) = Re\left\{e^{j2\pi f_c t}\sum_n \alpha_n(t)e^{-j2\pi f_c \tau_n(t)}s_l[t - \tau_n(t)]\right\} \tag{2.15}$$

The equivalent lowpass signal is

$$r_l(t) = \sum_n \alpha_n(t) e^{-j2\pi f_c \tau_n(t)} s_l[t - \tau_n(t) \tag{2.16}$$

As $r_l(t)$ is the response of the channel to the input signal of $s_l(t)$, it follows that the lowpass channel can be described by

$$c(\tau; t) = \sum_n \alpha_n(t) e^{-j2\pi f_c \tau_n(t)} \delta[\tau - \tau_n(t)] \tag{2.17}$$

This is an acceptable model for systems and environments in which the multiple paths are cleanly discretized, but this will not always be the case; more continuous multipaths are obtained by passing a signal through a fluid, such as a tropospheric scattering channel. In such a case, the sum could be replaced by an appropriate integral.

Depending the power distribution among the paths, they could be characterized by different models. There are two common stochastic models for multipath channels. The Rayleigh fading model is used when there is no line of sight signal, and so the received signal is a superposition of several copies with similar powers. In a Rayleigh fading model, the magnitude of the signal will fade according to a Rayleigh distribution. Due to the central limit theorem, this can in turn be modelled by a channel gain that is zero-mean and with the phase being evenly distributed between 0 and 2π radians.

In the event that there is a single dominant path, most likely the direct line of sight path, the Ricean fading model can be used instead. Multipath interference is still present, but even without explicitly accounting for the interference, proper decoding is

easier. The amplitude gain of a Ricean fading channel can be characterized by a Ricean distribution.

2.3 OFDM

OFDM, or Orthogonal Frequency Divison Multiplexing, has been used for the last several iterations of Wi-fi, since 802.11g, and is also used in modern cellular systems like LTE (Long-Term Evolution) and WiMAX (Worldwide Interoperability for Microwave Access). This section will briefly explore its characteristics and advantages over other modulation schemes.

2.3.1 Multicarrier Modulation

Due to the high cost of bandwidth, system designers strive to maximize the data throughput and bandwidth efficiency. This can be simple, particularly for narrowband channels, in which case there is little variation in the channel as a function of frequency. However, for a wideband channel, it could be difficult to utilize the full capacity due to the frequency-selective nature of the channel. Employing a single-carrier system in such an environment makes the waveform harder to decode at the receiver end due to the effects of time dispersion.

An alternative to this approach is multicarrier modulation, in which the bandwidth is subdivided into smaller parts. A fraction of the total message is then encoded and transmitted over each subcarrier. By thusly transforming a wideband system into several narrowband systems, equalization of the signal is simplified. Fig. 2.4, which shows the

frequency response of a channel divided into multiple subchannels. This approach yields transmission rates close to channel capacity, as shown in [15].

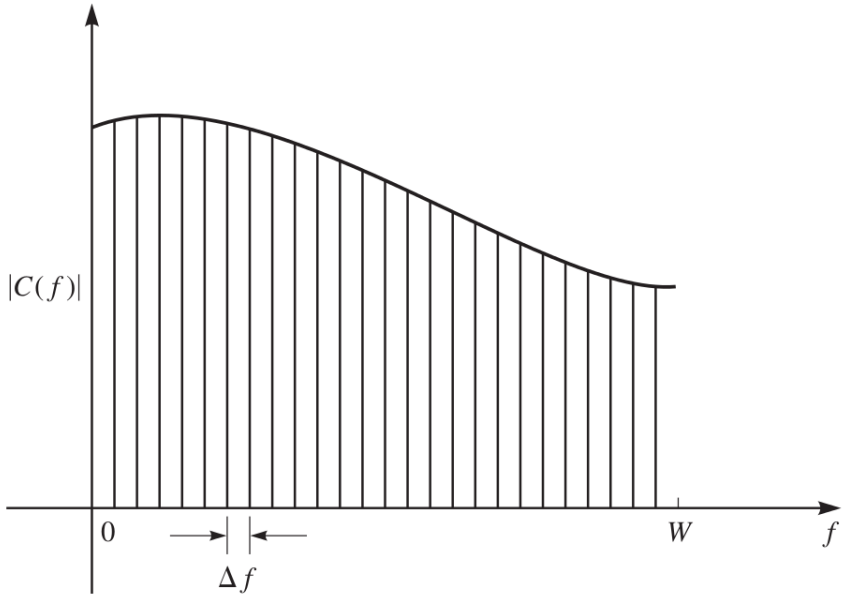

Figure 2.4: Wideband Spectrum Subdivided into Subcarriers [15, Fig. 11.2-2]

With a sufficiently large number of subcarriers, and, correspondingly, a sufficiently small bandwidth Δf allocated to each, the frequency response of the channel is constant for each subcarrier, which mitigates the effects of the channel in causing inter-symbol interference, or ISI.

2.3.2 Orthogonal Carriers

OFDM is a particular case of multicarrier modulation in which the subcarriers are orthogonal to each other. This approach means that, discarding negative channel effects, there is no interference between the signals transmitted along each subcarrier, termed inter-carrier interference, or ICI. In practice, doppler shifts caused by movement or other factors shift the subcarriers, destroying orthogonality and in effect reducing the SNR of

the signal. Fig. 2.5 shows the ideal frequency response of an OFDM transmission, with each peak representing a subcarrier. Note how each peak is located at a frequency where the other subcarrier waveforms are zero.

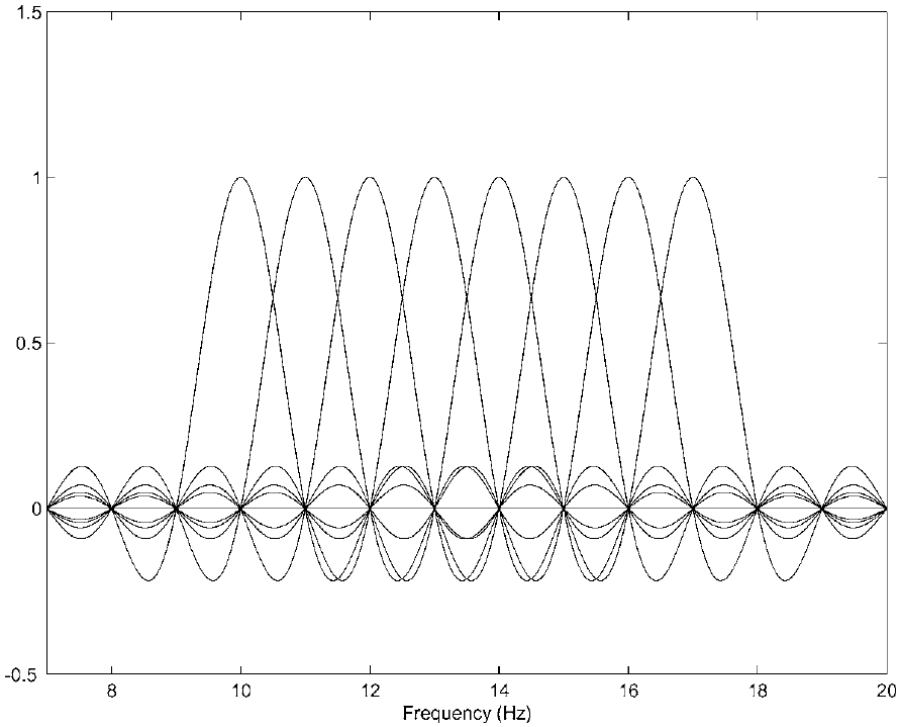

Figure 2.5: Orthogonality of OFDM Spectrum [14, Fig. 12.12]

This orthogonality is a major aspect of what made OFDM a popular modulation scheme within the past decade. It allows the subcarrier waveforms to be clustered closer together than competing multicarrier schemes, increasing bandwidth efficiency of the system.

Consider a generic multicarrier system consisting of k subcarriers. Within each subband, a sinusoidal carrier of the form

$$s_k(t) = cos 2\pi f_k t, \quad k = 0, 1, ..., N-1 \tag{2.18}$$

is transmitted. f_k here is the central frequency of the kth subchannel. If the symbol rate $\frac{1}{T}$ is equal to the frequency separation Δf of adjacent subcarriers, then the subcarriers can be shown to be orthogonal over the duration of the interval T, at least without channel corruption. This can be expressed as

$$\int_0^T cos(2\pi f_k t + \phi_k) cos(2\pi f_j t + \phi_j) dt = 0 \tag{2.19}$$

2.3.3 OFDM Transceiver Chain

Typical OFDM systems have many subsystems in common. The data are modulated in the frequency domain, converted from serial to parallel form to be distributed between available subcarriers, and converted to the time domain using an IDFT (Inverse Discrete Fourier Transform). A cyclic prefix of length μ is added to each subcarrier. The cyclic prefix ensures that as long as the impulse response of the channel is shorter than μ, the convolution operation inherent in passing data through the multipath channel becomes a cyclic convolution. The cyclic prefix eliminates inter-symbol interference, or ISI, between the data blocks; this is because the first μ samples of the channel output affected by ISI can be discarded without any information loss. [14, pg. 385] Thus, the cyclic prefix exhibits a trade-off of ISI versus throughput.

Following the cyclic prefix, the signal is converted into serial form, transformed into an analog signal, upsampled and transmitted. The receiver structure is the transmitter structure in reverse, as shown by Fig. 2.6.

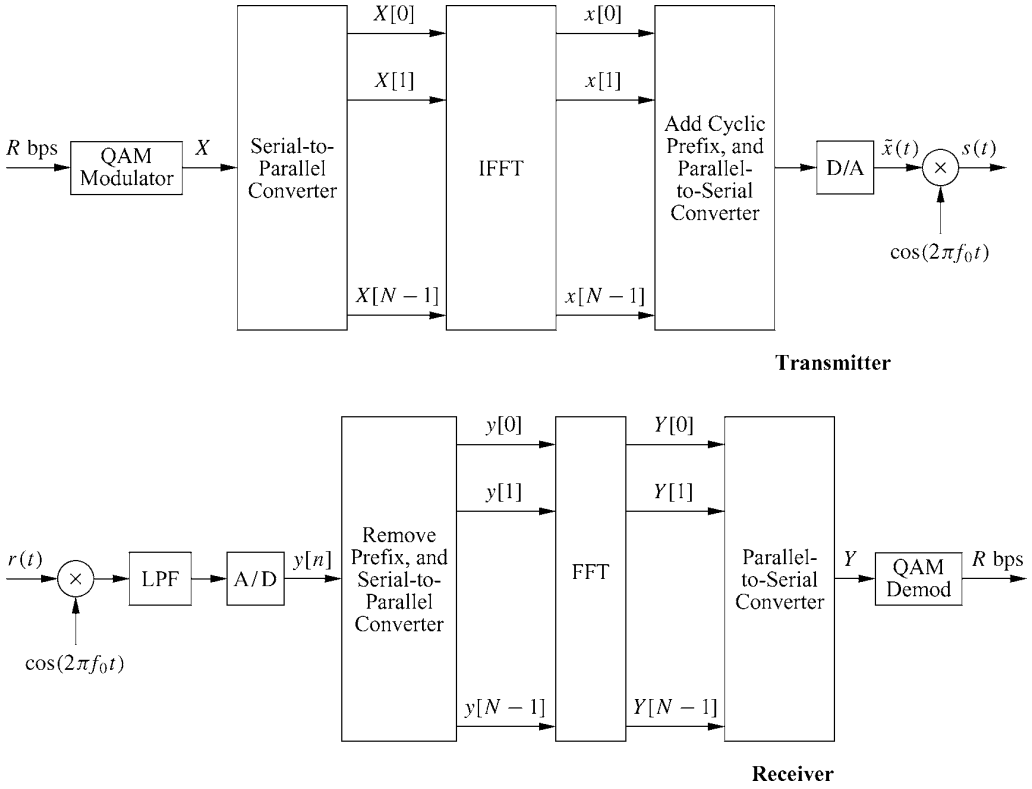

Figure 2.6: OFDM [14, Fig. 12.7]

Due to the transmission of the data in the time-domain following modulation in the frequency domain, OFDM allows for frequency-domain equalization. Unlike time-domain equalization, which necessitates inverting the impulse response of the channel (typically through the use of an adaptive filter or a similar approach), frequency-domain equalization entails inverting a single constant – the complicated time-domain impulse response can be characterized by a single multiplication by a complex constant in the frequency domain. This simplification is a major factor in the growing use of OFDM, as it greatly decreases the complexity, and thus cost, of receiver equalization. This assumption is central to the tested solution for joint detection and synchronization; though the derived genetic algorithm can be adapted to deal with time-domain equalization, that would decrease performance and increase computational complexity.

Many systems that use OFDM principles, such as modern cellular systems, use OFDMA, or Orthogonal Frequency Division Multiple Access. OFDMA uses the general principles of OFDM, but allows for intelligent allocation of subcarriers to different users. More modern improvements of this concept lead to carrier aggregation, wherein the transmitter can spread the information over several subcarriers and allocate several of those to individual users.

2.4 Diversity

Diversity is a technique in which the bit error rate of a transmission is reduced by relying on multiple signal paths. If one of them enters a deep fade, greatly lowering the SNR, the other path could be used instead; thus, communication could be made more reliable.

There are many diversity techniques available over different dimensions. Diversity over time can be as simple as repetition coding, in which the same signal is transmitted twice, or more complicated, as with coding and interleaving. In the case of a frequency-selective channel, diversity over frequency could be used; the same message could be transmitted over two or more different frequencies. Space diversity can be achieved by using multiple antennas to receive or transmit the signal.

The one factor binding these together is that their performance is maximized when the effects of the channel on the dimensions are independent. If, for example, spatially diverse transmission with two receive antennas is attempted but an object is blocking both, the transmission may still easily fail, due to the effects of the two channels not

being independent of each other.

Diversity techniques will be briefly discussed here to lead the reader into MIMO technologies, in which spatial diversity is expanded in such a way to potentially increase data throughput.

2.4.1 Space Diversity

Space diversity uses multiple transmit or receive antennas to improve the decoding performance. Systems with single transmitter antennas and multiple receive antennas are referred to as SIMO (Single-In Multiple-Out); systems with multiple transmitter antennas but a single receiver antenna are referred to as MISO (Multiple-In Single-Out).

Both of these techniques can greatly improve the error rate of a wireless communication system. This improvement can be described as an array gain and a diversity gain. Fig. 2.7 shows a basic SIMO receiver:

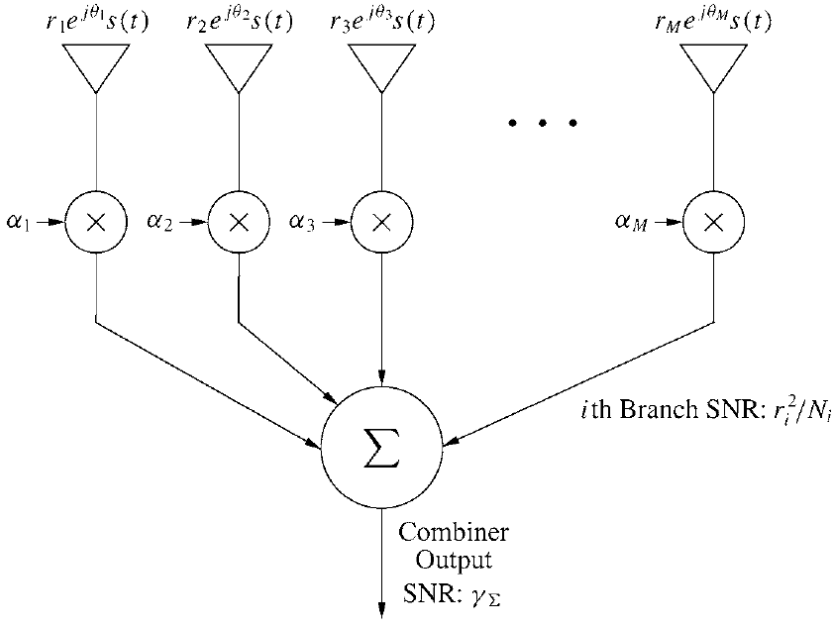

Figure 2.7: Linear Combiner for SIMO System [14, Fig. 7.1]

Array gains results from coherent combining of multiple signals. This improves performance even in the absence of fading. For example, assume that $r_i = \sqrt{E_s}$, where E_s is the energy per symbol of the signal. Assume that the noise power is $N_0/2$ on each branch, and pulse shaping such that $BT_S = 1$. In this case, each branch has the same SNR $\gamma_i = E_s/N_0$. Set the weights $a_i = r_i/\sqrt{N_0}$. These are the optimal weights for Maximal-Ratio Combining, a signal combination scheme in which each signal is weighed based on its SNR; in this case, due to all the signals having identical SNRs, the actual value of a_i does not matter as long as the weights are the same for all signals.

The received SNR can then be expressed as

$$\gamma_{sum} = \frac{\left(\sum_{i=1}^{M} a_i r_i\right)^2}{N_0 \sum_{i=1}^{M} a_i^2} = \frac{\left(\sum_{i=1}^{M} \frac{E_s}{\sqrt{N_0}}\right)^2}{N_0 \sum_{i=1}^{M} \frac{E_s}{N_0}} = \frac{M E_s}{N_0} \tag{2.20}$$

Thus, even in the absence of fading, there is an M fold increase in the SNR. [14, pp. 207] On a dB scale, this corresponds to a shift left of the BER curve, corresponding to the number of receive antennas. Array gain is formally defined as the ratio of the two SNRs: $A_g = \frac{\gamma_{sum}}{\gamma}$.

Diversity gain assumes independent fading, and yields a more impressive improvement: For some diversity systems, the probability of a symbol error can be expressed as $P_s = c\gamma^M$, where c is a constant depending on the system modulation scheme and M is referred to as the diversity order of the system. On the typical BER curve, this manifests as an increase of the slope of the BER/SNR plot. [14, pp. 208]

The maximum diversity order of a SIMO system is M, the number of receive antennas; when this is achieved, the system is said to achieve full diversity order. The same

follows with MISO systems. It should be noted that MISO is less prevalent than SIMO due to the requirement that all of the transmitting antennas maintain the same signal power to compete with a similar SIMO system; thus, more power is expended for similar performance.

2.4.2 Time Diversity

As mentioned previously, the simplest form of time diversity is a repetition code; the same symbol is transmitted several times along the signal stream. A more complicated application of this concept is the combination of forward error correction and interleaving. Forward error correction generates redundant data to add to the stream that would allow for full decoding at the receiver even if certain symbols are lost along the way.

Interleaving is used to maximize the independence of the 'channels' across which these symbols are transmitted. As an example, consider the repetition code. If every symbol is transmitted twice in quick succession, there is limited resistance in the scheme to deep fades, as these may last for several symbols at a time, and so some data could nevertheless be lost. By applying an interleaver to the repeating message, symbols representing the same information could be moved to different temporal positions, decreasing the chance that a deep fade would completely destroy the data.

More specific examples of forward error correction and interleaving will be examined in Chapter 4.

2.4.3 Space-Time Block Codes

Space-Time Block Codes, or STBCs, are codes that represent the transmission of symbols over the time and antenna dimensions. STBCs are usually presented in matrix format, where columns represent different transmit antennas and rows represent time-slots. Similarly to forward error correction schemes, STBCs have associated code rates which measure the number of symbols per time slot transmitted on average over the course of a block.

Probably the simplest and most effective STBC is the Alamouti code, also standardized in 802.11n and above as an optional transmission scheme [17, pp. 200]. The Alamouti code is used for 2x1 MISO configuration, and it is unique in the sense that it is the only STBC that achieves a diversity gain while maintaining a code rate of 1. The coding procedure for the Alamouti encodes the information of two symbols, s_1 and s_2, into a code word. The code word can be expressed by

$$\begin{bmatrix} s_1 & s_2 \\ -s_2* & s_1* \end{bmatrix} \tag{2.21}$$

This block code represents a simultaneous transmission over the two antennas of s_1 and $-s_2*$, followed by the simultaneous transmission of s_2 and s_1*. For a simple transmission, the received signal over the MISO channel is given by

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} h_1 & h_2 \end{bmatrix} \begin{bmatrix} s_1 & s_2 \\ -s_2* & s_1* \end{bmatrix} + \begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix} \tag{2.22}$$

where y_1 and y_2 represent the two received symbols, h_1 and h_2 represent the single-tap channel gains for the paths from both transmitters to the receiver, and Z_1 and Z_2 represent the noise added by the channel.

2.5 MIMO (Multiple-Input Multiple-Output)

MIMO takes advantage of the presence of multiple antennas to improve performance of the system. MIMO technology can be used for either diversity gain, which reduces the error rates but allows for the same data rate, or for the transmission of multiple data streams, increasing to the total data rate of the system. Note that a reduction in error rates could itself be translated to an increase in the data rate, as it would allow for more bits per symbol while maintaining similar BER. Fig. 2.8 shows a generic MIMO system with N_T transmitting antennas and N_R receiving antennas.

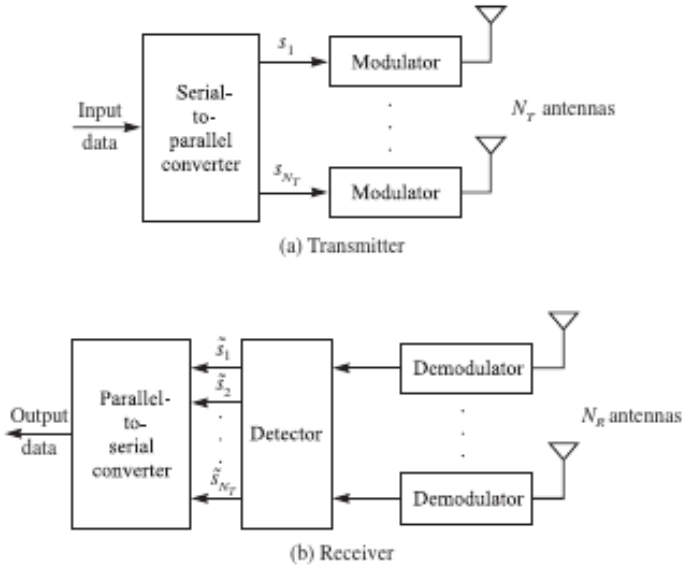

Figure 2.8: MIMO System [17, Fig. 7.5]

This section addresses the task of decoding a MIMO transmission where every

antenna is used for data transmission — no diversity is involved. This implies that $N_T = N_R$. Without diversity, very high SNRs are required in order to accurately decode all of the transmitted symbols; because of this, most commercially available systems that boast this feature — such as LTE or even Wifi — rarely use it.

This task is being discussed due to the similarities between this and the problem of joint detection. The joint detection problem can be considered an underdetermined variant of a typical MIMO problem, and so it's possible that some of the approaches to multiplexed MIMO may be modified and applied to a joint detection problem. This discussion will also give the reader a better feel for the entire problem of decoding interfering streams.

This section assumes that perfect synchronization has been performed; knowledge of $\boldsymbol{H}$, the channel matrix, is required, and no frequency or timing offsets are assumed. The received signal for this problem can be described by

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{z} \tag{2.23}$$

where $\boldsymbol{x}$ consists of the transmitted symbols and $\boldsymbol{z}$ represents the noise of the system. $\boldsymbol{y}$ and $\boldsymbol{z}$ are vectors N_R symbols long, $\boldsymbol{x}$ is a vector N_T symbols long, and $\boldsymbol{H}$ is a N_R by N_T channel gain matrix.

Several receiver architectures to solve for the transmitted signal $\boldsymbol{x}$ given knowledge of $\boldsymbol{H}$ and received signal $\boldsymbol{y}$ will now be explored.

2.5.1 Linear Decoders

Zero-Forcing

The Zero-Forcing algorithm, also known as the decorrelator, is the simplest MIMO receiver algorithm available. It attempts to find a matrix $\boldsymbol{W}$ that satisfies $\boldsymbol{WH} = \boldsymbol{I}$. It does this by inverting $\boldsymbol{H}$ with the Moore-Penrose pseudoinverse [18, pp. 351]. This yields

$$\boldsymbol{W} = (\boldsymbol{H}^{\boldsymbol{H}}\boldsymbol{H})^{-1}\boldsymbol{H}^{\boldsymbol{H}} \tag{2.24}$$

Minimum Mean Square Error

This simple approach is improved on with the Minimum Mean Square Error (MMSE) technique. The MMSE technique tries to find the $\boldsymbol{W}$ which minimizes $E\left(\left[\boldsymbol{Wy} - \boldsymbol{x}\right]\left[\boldsymbol{Wy} - \boldsymbol{x}\right]^{H}\right)$ [19, pp. 321]. This yields

$$\boldsymbol{W} = \left(\boldsymbol{H}^{\boldsymbol{H}}\boldsymbol{H} + N_0\boldsymbol{I}\right)^{-1}\boldsymbol{H}^{\boldsymbol{H}} \tag{2.25}$$

2.5.2 OSIC Signal Detection

The linear approaches to MIMO decoding, though computationally efficient and flexible, tend to be the least effective. OSIC, or ordered successive interference cancellation, utilizes the linear methods iteratively so as to improve performance [19, pp. 322]. The principle of ordered successive cancellation is that a linear decoder is used to decode

one of the spatial streams, at which point that decoded stream is subtracted from all of the other streams. The receiver then applies a linear decoder to the next stream. This cycle continues until all of the streams have been decoded. The 'Ordered' aspect of OSIC signifies that the order of the streams to be decoded is chosen to maximize performance. This is typically done by first decoding the streams with the highest SNR, or in the presence of an undecodable interferer, highest SINR. Such ordering significantly outperforms unordered SIC techniques.

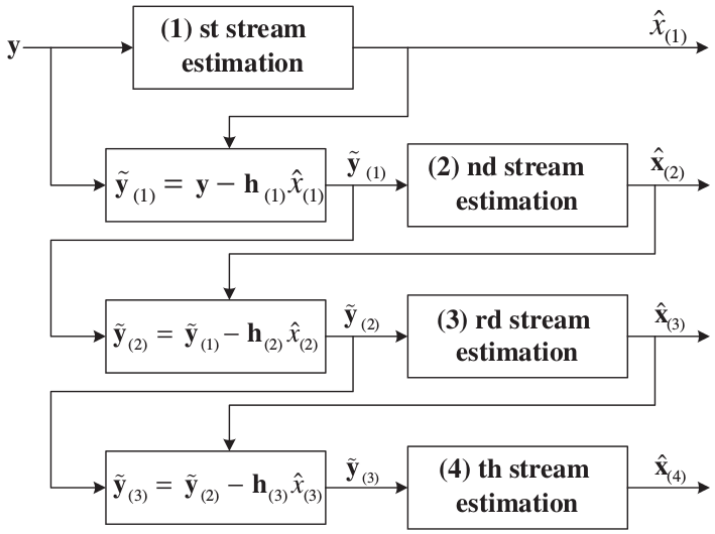

Figure 2.9: OSIC for 4 Spatial Streams [19, Fig. 11.2]

2.5.3 Near-ML Detection Schemes

ML, or Maximum Likelihood solutions to the problem of MIMO detection achieve optimal performance in the sense that the bit error rate is minimized. However, computational complexity could be easily simplified. This is very similar to the simplification done in [1] to reduce the derived JML (Joint Maximum Likelihood) to the JMD (Joint Minimum-Distance) detector. The minimum-distance detection scheme can be described

mathematically as

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x} \in C^{N_T}}{\arg\min} ||\boldsymbol{y} - \boldsymbol{H}\boldsymbol{x}||^2 \tag{2.26}$$

In plainer terms, this receiver minimizes the sum of the squares of the differences between the received and the calculated signal at each antenna. It necessitates an exhaustive search through all potential values of $\boldsymbol{x}$. Despite the massive simplification in evaluating the viability of each individual $\boldsymbol{x}$ combination relative to the true ML algorithm, this requirement for an exhaustive search is still computationally untenable. As the dimensionality of $\boldsymbol{x}$ grows — either due to the growth of the constellation, or the number of MIMO streams, or with multicarrier transmission schemes the number of subcarriers — the computational complexity grows exponentially.

There are a number of schemes which attempt to achieve the minimum-distance estimate by looking at a subset of all possible $\boldsymbol{x}$s.

QRM-MLD

One potential solution is the QRM-MLD (QR-decomposition M - Maximum Likelihood Decoder) method [19, pp. 329]. The remaining M represents the number of potential solutions that are kept by the algorithm when it progresses to the next stage of decoding.

The method relies on a QR decomposition of the channel matrix such that $\boldsymbol{H} = \boldsymbol{Q}\boldsymbol{R}$.

The minimum-distance metric can be expressed by

$$\begin{aligned} \| \boldsymbol{y} - \boldsymbol{H}\boldsymbol{x} \| &= \| \boldsymbol{y} - \boldsymbol{Q}\boldsymbol{R}\boldsymbol{x} \| \\ &= \| \boldsymbol{Q}^{\boldsymbol{H}}(\boldsymbol{y} - \boldsymbol{Q}\boldsymbol{R}\boldsymbol{x}) \| = \| \tilde{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{x} \| \end{aligned} \tag{2.27}$$

The matrix $\boldsymbol{R}$ is an upper triangular matrix, which can be used to simplify the decoding process. For this example, let $N_T = N_R = 2$, though the method really shines for higher dimensionalities, as computational complexity scales linearly with the number of spatial streams in the system. Then

$$\begin{aligned} |\tilde{\boldsymbol{y}} - \boldsymbol{R}\boldsymbol{x}|^2 &= \left| \begin{bmatrix} \tilde{y_1} \\ \tilde{y_2} \end{bmatrix} - \begin{bmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right|^2 \\ &= |\tilde{y_2} - r_{22}x_2|^2 + |\tilde{y_3} - r_{11}x_1 - r_{12}x_2|^2 \end{aligned} \tag{2.28}$$

The QRM-MLD algorithm attempts to iteratively minimize each term in Eqn. 2.28. First, M values of x_2 out of the constellation that minimize the error $|\tilde{y_2} - r_{22}x_2|^2$ are chosen. For each of these most likely values, the M values of x_1 of the constellation that minimize $|\tilde{y_2} - r_{22}x_2|^2 + |\tilde{y_3} - r_{11}x_1 - r_{12}x_2|^2$ are chosen. If N_T and N_R were higher than two, this process would continue, with only M solutions being carried over into each subsequent stage.

The performance of the QRM-MLD algorithm is very dependent on the value of M — the higher it is, the better the performance, though the more computationally complex the algorithm is. Additionally, note that this algorithm requires a constant number of operations, making it an attractive option for implementation in hardware.

Sphere Decoder

The sphere decoding method, in essence, manages to consider only a small subset of the potential solutions by maintaining a 'sphere' of varying radius [19, pp. 329]. The algorithm adjusts the radius of the sphere in the solution space until a single solution is encased.

Figure 2.10: Sphere Demonstration [19, Fig. 11.5]

Note that the Sphere Decoder and the QRM-MLD algorithms take opposite approaches to the problem of finding the minimum-distance solution. The Sphere Decoder guarantees finding it, but requires a varying — but typically much smaller than the time it takes to do a full exhaustive search — period of time. The QRM-MLD algorithm, by contrast, does not guarantee finding it, and the variable M can be modified to meet computational complexity needs.

2.5.4 Relative Decoder Performance

Fig. 2.11 compares the performance of the techniques discussed in this section.

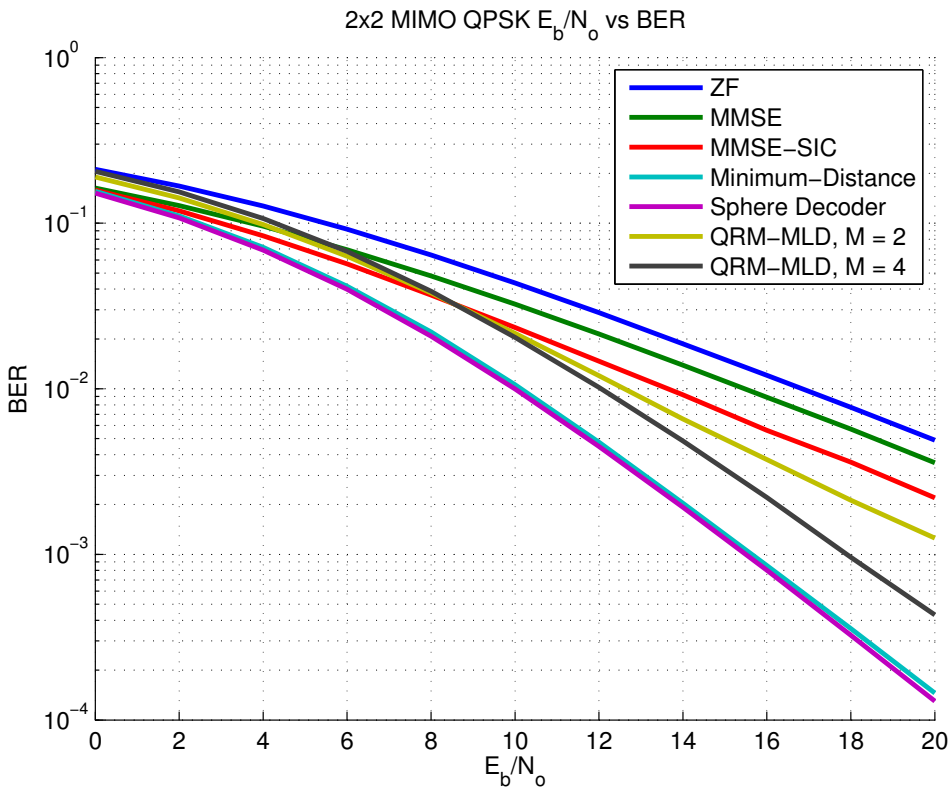

Figure 2.11: Comparison of MIMO Decoding Performance

The linear MIMO decoders are the worst performers, while adding ordered successive interference cancellation to MMSE significantly improves performance. The performance of the QRM-MLD algorithm depends on the number of solutions stored per iteration, or M; at M = 2, performance is scarcely superior to MMSE-SIC, whereas for M = 4, the performance is approaching the optimal Minimum-Distance decoder.

It was considered that the sphere decoder or QRM-MLD algorithms could be adapted for the underdetermined MIMO case – with fewer receiver antennas than there are spatial streams. This could prove an attractive alternative to the JMD algorithm, discussed in more detail later on, which corresponds very closely to the near-ML exhaustive search MIMO detector. This is left to future work.

Chapter 3

The 802.11 Standard

802.11 is is a set of media access control (MAC) and physical layer (PHY) specifications for implementing wireless local area network (LAN). This section will describe the basics of the standard to familiarize the reader with the technology.

3.1 PHY Layer

The latest iterations of the 802.11 standards — 802.11a, 802.11g and later — use OFDM modulation with 64 subcarriers. Of these, only 48 are used for data transmission. The outer 12 subcarriers are zeroed out, in order to insulate the data stream from interference in nearby frequency bands. Four subcarriers are reserved for pilot signals, which are continuously used by the receiver for synchronization, like channel estimation [14, pp. 397].

Modern Wifi standards operate at three ISM frequency bands. ISM bands — standing for industrial, scientific and medical radio bands — are portions of the spectrum

reserved internationally to allow free use below certain signal powers. The most popular Wifi band is at 2.4GHz, used by the 802.11g, 802.11n and 802.11ac standards. The 5GHz band is more sparsely populated, but is still in use by the 802.11a, 802.11n and 802.11ac standards. A relatively new addition to Wifi is the 60GHz 802.11ad standard. The 802.11ad standard was developed due to a desire to massively increase the potential data rate of WLAN systems. The downside to working in the 60GHz band is the higher susceptibility of the waves to shadowing, leading to higher attenuation during transmission.

3.1.1 Forward Error Correction

The 802.11 standard uses convolutional encoding for Forward Error Correction, or FEC. The purpose of FEC is to take the stream of bits before transmission and output a stream with the same data, but with more resilience to loss. This necessitates increasing the length of the transmission, reducing data rates, but the large performance improvements with regard to bit error rate are deemed sufficient to justify the trade-off.

The process used to generate the output data stream is shown in Fig. 3.1. For every bit of input data, two output bits are generated, making this a rate 1/2 encoder. Note that convolutional encoders force delay into the system, as it takes time for the data to work its way through the encoder.

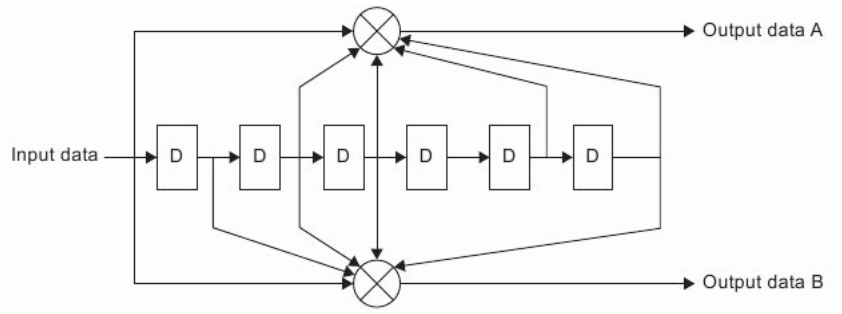

Figure 3.1: IEEE 802.11 a/n rate-1/2 convolutional encoder [17, Fig. 7.4]

The 802.11 standards have adaptive modulation schemes. This means that they can adjust their modulation schemes — BPSK, 4-QAM, 16-QAM, 64-QAM, and even 256-QAM in the newer standards — as well as the encoding schemes to achieve certain trade-offs between throughput and reliability. The standard specifies that changes in the encoding rate — switching from 1/2 to 3/4, for example — is done by puncturing the data stream output by the rate 1/2 convolutional encoder. In this context, puncturing means removing the bits from the data stream. The pattern by which the data is punctured is standardized and specified in the preamble of the frame, allowing the receiver to flag those 'missing' bits as unknown and perform Viterbi decoding.

By removing the bits, more of the stream can be allocated to transmit new data, increasing data rate. At the same time, by decreasing the encoding rate, the system is made more vulnerable to the effects of the channel. Fig. 3.2 demonstrates the process of puncturing.

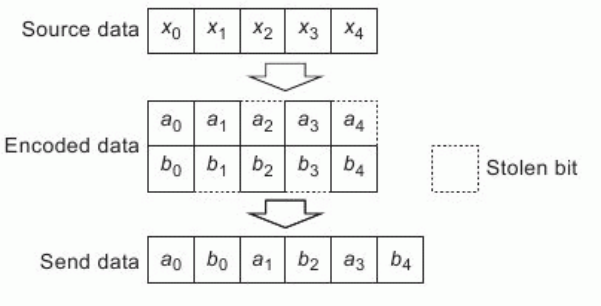

Figure 3.2: Puncturing for coding rate 5/6 [17, Fig. 7.5]

3.1.2 Interleaving

Interleaving is used in the 802.11 standard due to the weakness of convolutional encoding in the presence of a stream of bit errors in the channel. The 802.11 standard uses a particular type of interleaving called block interleaving. Block interleaving is very simple; in essence, it reshapes the data stream into a rectangular matrix, and returns the reading of the transpose of that matrix.

3.1.3 MIMO

802.11 standards use MIMO technology, or spatial multiplexing. MIMO was first introduced in Wifi in the 802.11n standard, with MIMO up to 4x4. The 802.11ac standard increased the maximum MIMO to eight streams (8x8). In practice, these are not necessarily supported by all receivers; in fact, very few 802.11ac receivers support the full eight spatial streams. Even when they are fully supported, they are rarely used due to the difficulty in properly synchronizing the transmission; very high SNR is required. Using the antennas for diversity gain is a more common practice, due to the relative computational simplicity. The 802.11 standards also support STBCs, such as the Alamouti

code and several others.

3.2 MAC Layer

Wifi uses the CSMA MAC protocol, or Carrier-Sense Multiple Access. The transmitter "listens" to the radio spectrum it wants to transmit over, trying to determine whether another transmitter is currently occupying it. The Multiple Access refers to the fact that multiple nodes transmit over the same bandwidth.

Wifi uses a specialized variant of CSMA called CSMA/CA, where the CA refers to "collision avoidance". This attempts to solve the problem of two transmitters noticing that the channel is open simultaneously, and thus both transmitting a frame and interfering with each other. CSMA/CA introduces a contention period — a random period of time that the transmitter waits before beginning transmission. If, in that period, the channel remains clear of other signals, then transmission goes forward; if not, then the transmitter waits for another opening.

If a data frame is transmitted and no acknowledgement is received, then after a random period of time, provided that the channel is free, the frame is retransmitted.

3.2.1 Hidden Node Problem

The hidden node problem describes one of the situations in which the CSMA/CA system fails. In essence, the CSMA system allows the transmitter to listen for an interferer at its own location, so it can use that as an estimate for the presence of a signal at the receiver. However, if there is a secondary transmitter further away from the transmitter

than from the receiver, then the transmitter could decide that the spectrum was clear and that that it was safe to transmit, even though the receiver is receiving another frame. This situation is shown in Fig. 3.3.

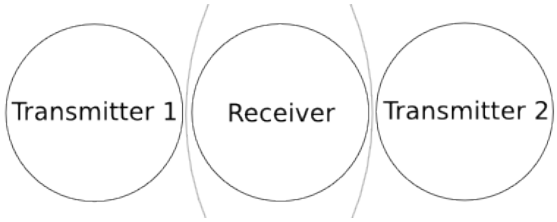

Figure 3.3: Hidden Node Problem

A related problem to the hidden node problem is the exposed node problem. Whereas the hidden node problem results in transmitter sending when there is an interferer present at the receiver, the exposed node problem results in a transmitter holding back transmission despite the spectrum at the location of the receiver being empty. This is caused by an interfer located closer to the transmitter than to the receiver.

Collision Avoidance

RTS/CTS (Request-To-Send/Clear-To-Send) is a mechanism developed to solve the hidden node problem (but not, necessarily, the exposed node problem). With RTS/CTS enabled, the transmitters send out a short RTS frame, asking the receiver whether the spectrum is free for a large data frame. If and when it is, the receiver transmits a CTS frame, letting the transmitter know that there are no interferers.

The RTS/CTS mechanism does not manage to solve the exposed node problem like it does the hidden node due to the fact that if the transmitter sees interference, no RTS frame is sent in the first place; thus, no knowledge is transmitted back to the transmitter

from the receiver in the form of a CTS. However, if the nodes are synchronized with each other, then they could use knowledge of other nodes RTS/CTS frames to decide whether or not they are exposed nodes. If a node hears an RTS from a neighboring node but no CTS, the node can deduce that it is an exposed node.

Chapter 4

Stochastic Optimization Algorithms

The problem of joint detection and synchronization was formulated as an optimization problem over many dimensions, and a stochastic optimization algorithm was used to perform the optimization. Stochastic optimization algorithms use random variables in the process of searching for the optimal solution. Stochastic optimization algorithms were chosen over deterministic algorithms due to the non-convexity of the solution space; a deterministic gradient descent or similar approach would have proven insufficient to converge to an acceptable answer.

In general, stochastic optimization functions are useful for solving the following class of problems: "Find the value(s) of a vector $\boldsymbol{\theta} \in \Theta$ that minimize a scalar-valued *loss function* $L(\boldsymbol{\theta})$" [13, pp. 2]. Vector $\boldsymbol{\theta}$ represents a vector of variables that the user is trying to optimize. The loss function $L(\boldsymbol{\theta})$ is a scalar measures that allows for the stochastic algorithms to compare the "usefulness" of the solutions. When stochastic optimization algorithms are applied, for example, to a financial problem, a fitting loss function would

be the inverse of the expected profit, so that minimizing it would maximize the profit. $\boldsymbol{\theta}$ can include any variables that the cost function could combine into comparable values. For the problems discussed here, $\boldsymbol{\theta}$ will be restricted to numbers both discrete and continuous. The solution space of the loss function $L(\boldsymbol{\theta})$, thus, may be continuous in some dimensions and discrete in others.

One of the major distinctions in optimization is between local and global optimization. Local optimization looks for any solution in $\boldsymbol{\theta}$ where $L(\boldsymbol{\theta})$ is minimized relative to its immediate vicinity of solutions $\boldsymbol{\theta}$. Global optimization, on the other hand, looks for the $\boldsymbol{\theta}$ that minimizes $L(\boldsymbol{\theta})$ over the entire solution space. Global optimization is a much more difficult problem than local optimization, as a simple gradient approach technique, which uses the "slope" of the solution space to determine the direction in which the minimum is, will not necessarily find the global minimum.

The potential difficulty of finding a global optimum solution is demonstrated in Fig. 4.1 below. This is a one-dimensional optimization problem in a continuous scalar value θ. In the easy problem, there is a local minimum on the left, and a global (and local) minimum on the right of the solution space. Depending on where the optimization algorithm begins, a simple gradient descent approach may yield the global optimum. In either case, many non-gradient based approaches may yield the global optimum. However, in the hard case, the global minimum of $L(\theta)$ is a single value. This is essentially impossible to find with any optimization algorithm; the "slope" of the surrounding area yields no clue that there is a global maximum present, and the chances of a numerical algorithm accidentally finding the correct global minimum are infinitesimal. Nevertheless,

the local minimum could be found with ease.

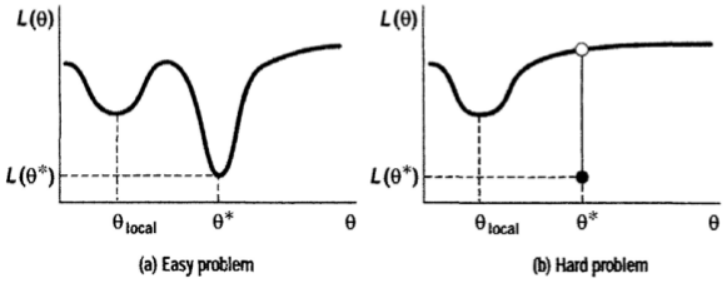

Figure 4.1: Example of an easy and hard problem for global optimization. [13, Fig. 1.2]

4.1 Key Points

Before proceeding with a brief overview of several stochastic optimization algorithms, there are several points about these algorithms that must be made. These are selected based on perceived relevance from [13, pp. 12-18].

4.1.1 Measuring Algorithm Efficiency

There are many different ways to measure the efficiency of stochastic optimization algorithms. Many of them rely on iterations, so one can count the number of iterations; all of them take a finite amount of time to run, so that could be used. However, both of these are flawed measures; the amount of "work" done per iteration could vary greatly between different algorithms. The time for something to run depends greatly on the operating system, language and processor architecture; what may run slowly on a standard computer as a MATLAB simulation may be very quick to run when implemented in a lower-level programming language like C or when implemented on dedicated hardware.

A more objective measure of algorithm efficiency is the number of cost function evaluations that need be done. These evaluations tend to make up the greater part of the computational complexity of an algorithm. This measure, however, is also not sufficient to be used as the end-all for algorithm judgement. One reason this is so is because different evaluations of $L(\boldsymbol{\theta})$ could be worth a different amount depending on the algorithm. This is most clearly demonstrated when considering the effects of parallelization. Several stochastic optimization algorithms allow for a population of solutions to be considered simultaneously, which opens the door to simpler, more efficient implementation in hardware. Thus, it is insufficient to simply look at the number of cost function evaluations; their effect on feasibility of utilizing the algorithm must also be considered.

4.1.2 Noise Robustness

Any application of stochastic optimization for implementing joint detection would necessarily require dealing with noise, which would be generated by all parts of the system: the channel, the temperature-induced noise of the RF components, etc. For the case of AWGN noise, error decreases at a rate of $\frac{1}{\sqrt{N}}$ [13, pp. 14], where N represents the number of measurements. As such, it is impossible to completely remove the effects of noise, and any attempts to mitigate it exponentially increase the computational complexity of the algorithm.

4.1.3 Curse of Dimensionality

As the length of vector $\boldsymbol{\theta}$ grows and the range of possible values that could be taken by the variables increases, the size of the solution space grows exponentially [13, pp. 14]. As such, the algorithms for high-dimensionality problems must scale well with dimensionality.

4.1.4 Measuring Convergence

With stochastic optimization problems, there is the problem of knowing when the solution is good enough to cease iterating and looking for a new solution. There is no simple, obvious way of doing it that would guarantee a certain level of performance; for nontrivial problems, there will always be an unexplored area in the solution space for any finite number of iterations, and so there is a non-zero chance that the global optimum solution, $\boldsymbol{\theta}^*$, lies within that region.

4.2 Algorithms

This section will describe several basic stochastic optimization algorithms, before explaining why the genetic algorithm was chosen for this problem.

4.2.1 Random Search

A basic stochastic optimization algorithm is a blind random search. First, the algorithm initializes; a beginning value for $\boldsymbol{\theta}$ is selected. Without any prior information,

this value should be randomly selected from a uniform distribution over $\boldsymbol{\Theta}$, the solution space. The value of $L(\boldsymbol{\theta})$ is calculated. from then on, the algorithm will randomly pick other solutions from the same distribution and compare the value of $L(\boldsymbol{\theta})$. Whenever the new $L(\boldsymbol{\theta})$ is smaller than the old, the stored solution is updated with the new solution.

The algorithm could be stopped based on a total number of iterations, which makes the convergence time of the algorithm predictable and which makes it simpler to implement in hardware. Alternatively, the algorithm could check every iteration whether a convergence condition was met.

In most ways, this seems like an overly simplistic algorithm, yet in a very unpredictable solution space, its random selections from the solution space may be scarcely less efficient than the informed selections of more sophisticated algorithms.

4.2.2 Simulated Annealing

Simulated Annealing is a stochastic optimization algorithm inspired by the natural process of annealing, which happens as a liquid or a solid cools. As temperature decreases, the molecules in a material lose mobility and molecules *may* tend to align themselves in a crystalline structure. The crystalline structure is the minimum-energy of the state, but note that it will not necessarily be reached; the cooling must occur at a sufficiently slow rate [13, pp. 209]. Without sufficiently slow cooling rate, the substance settles into a polycrystalline state that is not at its minimal energy.

This principle can be applied to stochastic optimization; this is called Simulated Annealing (SAN). The minimization of the cost function is analogous to the minimization

of the system energy. Unlike the random search algorithm and many others, which only allow for updates that lower the value of the cost function, Simulated Annealing can accept increases in $L(\boldsymbol{\theta})$ as it acquires enough information to find the global minimum. SAN bases this behavior on the Boltzmann-Gibbs probability distribution of statistical mechanics. Based on that, the probability of a system having a particular discrete energy state is given by:

$$P(x) = c_T exp\left(-\frac{x}{c_b T}\right) \tag{4.1}$$

where x is the energy state and T is the temperature. $c_T > 0$ is a normalizing constant, and $c_b > 0$ is Boltzmann constant [13, pp. 209]. Thus, as T decreases, the probability of the system being at a higher energy state decreases.

Each iteration, SAN generates a new value to solution $\boldsymbol{\theta}$ to test against the current value. If the new $L(\boldsymbol{\theta})$ is lower than the old, then the old $\boldsymbol{\theta}$ is replaced; however, if the new is higher, than the replacement happens under a condition known as the Metropolis Criterion. The algorithm can be formally explained thusly:

Let $\delta = L(\boldsymbol{\theta_{new}}) - L(\boldsymbol{\theta_{old}})$. If $\delta < 0$, then $\boldsymbol{\theta_{new}}$ is accepted; otherwise, it is accepted only if a uniform (0, 1) random variable U satisfies $U \leq exp\left(-\frac{\delta}{c_b T}\right)$. This is done a certain number of times until an equilibrium is reached, at which point the temperature T of the system is decreased. As with random search, the SAN algorithm could be stopped after a certain number of iterations or once some conditions have been met.

4.2.3 Genetic Algorithms

Genetic algorithms (GEs) fall under a class of algorithms that mimic the process of natural evolution known as evolutionary computation [13, pp. 231]. Unlike the two previously discussed optimization algorithms, genetic algorithms work with populations of solutions, not single solutions. This was a large factor towards selecting a genetic algorithm for implementation in this project; working directly with large groups of solutions allows a parallelization of the computation of $L(\boldsymbol{\theta})$ every iteration, greatly increasing the feasibility of implementing the algorithm in dedicated hardware.

As with simulated annealing, genetic algorithms are described in terms similar to those of their inspiration — in this case, evolution. Specific values of $\boldsymbol{\theta}$ are referred to as chromosomes [13, pp. 234], whereas the individual variables θ within $\boldsymbol{\theta}$ are referred to as genes. A p-dimensional $\boldsymbol{\theta}$ thus corresponds to p genes per chromosome. Evolutionary terms are also used to describe the operations performed to generate a new set of solutions (or chromosomes) every iteration. The genetic algorithm performs inheritance, mutation and crossover operations on the old population to generate a new population.

Inheritance refers to the direct copying of individual genes from solutions in the old population. Mutation refers to generating a new value for a gene to introduce new information into the population. This is useful in the event that the old population did not have the proper genes for the genetic algorithm to properly converge; convergence in that case could be impossible. Crossover refers to two chromosomes combining genes to generate a chromosome in the new population. These operations are performed on the more fit (smaller $L(\boldsymbol{\theta})$) members of the older population, to improve the chance of the

new population being more fit than the older.

There are many ways to implement a genetic algorithm. Apart from the basics, as described above, there are many ways to modify its behavior in order to best suit your purposes. The way the next population is generated can be modified, as the user could elect to either create a large temporary population and pick the best for the next iteration, or simply to select a few chromosomes at random and combine them. The algorithm could be elitist, meaning that it could deterministically place the most fit chromosomes from the old population into the new, or that operation, too, could be a random process.

There are some limits to what genetic algorithms can do, however; they are not the best choice for all stochastic optimization problems. For example, genetic algorithms rely on the different genes being independent. If they are not independent, then the efficacy of the cross-over operation is reduced, and convergence takes longer.

Chapter 5

Implementation

This paper considers the detection and decoding of an interferer while striving to maintain acceptable BER for the primary frame. Prior to the arrival of the interferer, the received signal $y[n]$ at time n is given by

$$y[n] = h_1 x_1[n] + z[n] \tag{5.1}$$

where $x_1[n]$ is the primary transmitted symbol at time n and h_1 is the channel gain from the transmitter to the receiver. $z[n]$ is the AWGN at receiver with variance of $N_0/2$ per dimension. The transmitted symbols $x_1[n]$ are from discrete constellations such as phase-shift keying (PSK) or quadrature-amplitude modulation (QAM). A quasi-static flat fading channel is assumed, allowing h_1 to remain constant over all n.

Once the interfering frame arrives, the received signal $y_2[n]$ at time n is given by

$$y[n] = h_1 x_1[n] + h_2 r[n] x_2[n] + z[n] \tag{5.2}$$

As with h_1 and $x_1[n]$, it is assumed that h_2 is constant, and $x_2[n]$ is from a discrete constellation. $r[n]$ is a constellation rotation factor of the second transmitter relative to the second, caused by oscillator frequency offsets or movement-induced Doppler shifts.

During the period of the collision when $x_2[n]$ consists of training symbols, this knowledge is exploited to allow the system to estimate h_2 and r. After the training symbols, the interfering frame contains data, and the system switches to a joint detection algorithm that uses knowledge of h_1, h_2 and r to estimate x_1 and x_2. For the purpose of this paper, it is assumed that the interferer arrives at the receiver at some point after the primary frame has been synchronized, but before the data portion of the frame is completed. This assumption covers the majority of collisions, due to the length of the data portion of a Wifi frame exceeding that of the preamble [12].

5.1 Interferer Detection

The first part of the system determines the presence of an interferer during the decoding of a primary frame. If an interferer is detected, the time t at which it appears at the receiver is estimated.

Three approaches to detection were developed that relied on different amounts of information being available at the receiver. All of them assumed that no phase information about the interfering frame is available; however, one of the three assumed full prior knowledge of both the SIR and the SNR, and another removed the dependence on the SIR by assuming prior knowledge of the distribution of the SIR in the system.

All three systems determine whether an interferer is present by looking at the error

magnitude after demodulation of each received symbol corresponding to the primary frame. The algorithms use this information differently.

5.1.1 No Prior Knowledge

This simplistic scheme assumes nothing about the SNR or SIR. In essence, the algorithm assumes that when the interferer appears, the error magnitude will increase. The algorithm finds the symbol at which the difference between the mean of the error magnitude before and after is maximized. This can be expressed by

$$d_n = \left(\frac{1}{n-1} \sum_{i=1}^{n-1} \epsilon_i - \frac{1}{N-n+1} \sum_{i=n}^{N} \epsilon_i \right)^2 \tag{5.3}$$

where d_n is the difference at time n, ϵ_i is the error magnitude at time i, and N is expected length of the interfering frame. The time n where d_n is the best estimate is decided to be the beginning of the interference. The presence of an interferer could be judged by thresholding d_n; if it exceeds a certain value, then an interferer is likely.

5.1.2 SNR and SIR Prior Knowledge

This algorithm assumes full knowledge of interferer SIR and SNR. It looks at the error magnitude of each symbol, and compares that to the expected error magnitude given knowledge of the SNR and SIR. For example, if an interferer arrives at time n, it is expected that the error magnitude would tend towards the expected error magnitude of a stream with no interference for symbols arriving prior to n, and towards that with

interference at time n and after. The point in time that minimizes the mean-squared of the difference between error magnitudes over all times n gives a good estimate for the beginning of the interfering frame. If no interferer is present, then the mean-squared error would be minimized under the assumption of no interference for the observed duration. The mean-squared error corresponding to the assumption of no interferer is given by

$$e = \frac{1}{N}\sum_{i=1}^{N}(\epsilon_i - \epsilon_{NI})^2 \tag{5.4}$$

where N is the expected length of the interfering frame, ϵ_i is the error magnitude after demodulation at time i, and ϵ_{NI} is the expected error magnitude for the current SNR, SIR and constellation without an interferer.

The receiver must also test the hypotheses that the interferer arrives at during one of the symbols of the primary frame. The error corresponding to these at time $n \in [1, N]$ is given by

$$e_n = \frac{1}{N}\left(\sum_{i=1}^{n-1}(\epsilon_i - \epsilon_{NI})^2 + \sum_{i=n}^{N}(\epsilon_i - \epsilon_I)^2\right) \tag{5.5}$$

where e_n is the error for time n and ϵ_I is the expected error magnitude with an interferer.

If the case where no interferer is present does not minimize the error, then the the n that minimizes e_n is decided to be the symbol at which the interfering frame first arrives.

The expected values for ϵ_{NI} and ϵ_I were derived by simulation, and the difference between them determines the efficacy of the algorithm; a larger difference increases the algorithm's resilience to noise. The values would be stored at the receiver. This a

negligible cost; the values ϵ_{NI} and ϵ_I are not particularly volatile with respect to SIR or SNR, and so a resolution of 1dB or so with regards to both would allow for reasonable detection performance.

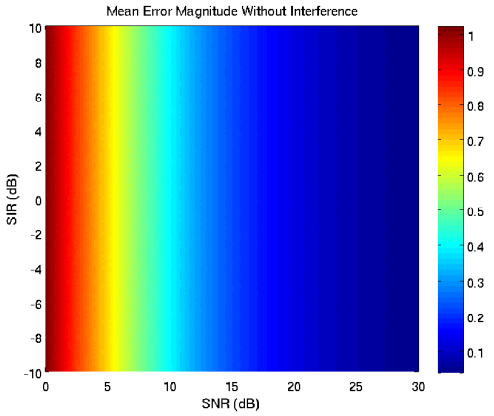

Figure 5.1: Expected error magnitude without interference

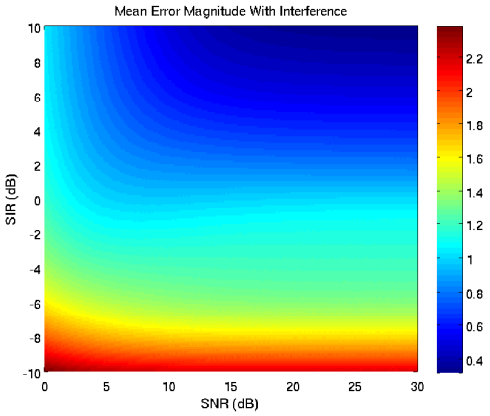

Figure 5.2: Expected error magnitude with interference

Fig. 5.3 shows the difference between the two error magnitudes. The larger the difference between the two, the better this algorithm can distinguish between the two cases, and the better it will perform at determining whether the interferer is present, and if it is, when it appears.

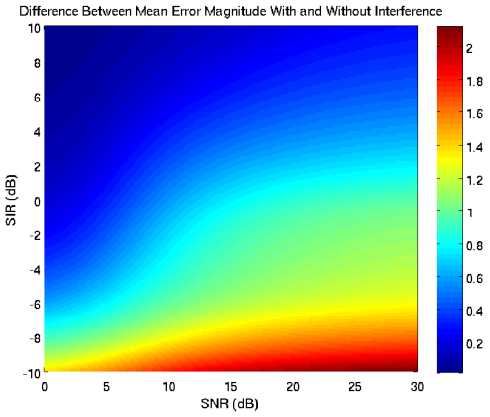

Figure 5.3: Difference between the expected error magnitude for the cases with and without interference

As it is, this scheme is heavily weighed towards the possibility of a false alarm of the detection, rather than towards a false negative; this is caused by every iteration of this algorithm testing many possibilities, of which no interference is only one. This can be alleviated by introducing a threshold during the comparison of the e and the e_ns. This is done in Chapter 6 to generate ROC

(Receiver Operating Characteristic) plots.

5.1.3 SNR Prior Knowledge

This algorithm performs the same computation as the previous algorithm that assumes full knowledge of SNR and SIR, but it assumes a probabalistic distribution of SIR. For the default implementations in this paper, it was assumed that the SIR was uniformly distributed from –6 dB to 6 dB. Instead of the thresholding values being drawn from a specified SIR and SNR, thus, they were drawn only from a specified SNR, with the expected error magnitude being the mean of the expected error magnitude from –6 dB to 6 dB. The implementation used here simply averaged the expected error magnitudes over the SIRs; a more effective implementation could perhaps estimate the probability of the interferer being present for each SIR, and average that over the SIRs.

5.2 Synchronization

The synchronization step occurs once the detection algorithm determines the delay relative to the primary frame at which the interfering frame first arrived at the receiver. The received signal during this stage is given by Eqn. 5.2. It is assumed that the receiver has already synchronized the initial signal and that the second frame is transmitting its training sequence, so h_1 and $x_2[n]$ are known. It is also assumed that the receiver has knowledge of the modulation formats of both signals.

The first step to enable the use of an optimization algorithm is to formulate the problem in such a way that there is a simple cost function to minimize. This allows

solutions such as genetic algorithms to be applied to the problem; the algorithms can judge the potential usefuless of solutions relative to one another. The cost function developed for this problem is given by

$$J = \sum_{n=1}^{N} \left(y[n] - h_1 x_1[n] - h_2 x_2[n] r[n] \right)^2 \tag{5.6}$$

where N represents the length of the training sequence. The cost function here represents the sum of the squares of the error magnitudes over the training symbols during which the joint detection and synchronization is occuring.

Making the assumption that the frequency offset between the two transmitters remains constant for the duration of the frame and that the duration of each symbol is the same, the $r[n]$ term can be replaced with $e^{jn\theta}$, with θ representing the angle of rotation during the period of a single symbol. The relationship between θ and the actual frequency offset in Hertz is given by

$$f_{error} = \frac{\theta}{2\pi T_s} \tag{5.7}$$

where T_s is the symbol rate of the system. The cost function can then be reformulated as

$$J = \sum_{n=1}^{N} \left(y[n] - h_1 x_1[n] - h_2 x_2[n] e^{jn\theta} \right)^2 \tag{5.8}$$

For the case where $N = 12$, as in 802.11 frames, this is a 14-dimensional optimization problem in h_2, θ and $\boldsymbol{x_1}$.

A genetic algorithm was chosen to optimize the given cost function. Relative to

other optimization techniques, genetic algorithms are particularly well-suited for solving the problem at hand due to the problem's non-convexity, non-linearity, and the fact that the algorithm does not necessitate calculating the gradient in the solution space. However, the genetic algorithm performs best when the variables of the chromosome $\boldsymbol{\theta}$ are independent. The twelve symbols $\boldsymbol{x_1}$ are completely independent, with their effects being incorporated into the cost function in quadrature with each other. However, the effects of θ and h_2 on the cost function are not independent from $\boldsymbol{x_1}$, leading to some wasted potential in the optimization.

5.2.1 Application

During the initialization of the genetic algorithm, a population of m potential solutions is generated with random potential values for these variables. The ideal m for effective convergence depends on the constellation size of the primary frame and dimensionality of the problem at hand; as either grow, m should increase. Any prior knowledge about the distribution can be incorporated during initialization to accelerate convergence. For example, if it is known beforehand that there is a maximum frequency offset, the initialization for θ can limit the possible values. If no prior information exists, $\theta \sim \mathcal{U}(0, 2\pi)$ can be used. Let $\boldsymbol{K}$ be the vector of m potential solutions.

After the initialization, a cross-over matrix $\boldsymbol{M}$ is generated, where $M_{OU} = f(K(O), K(U))$.

$f()$ is here a function that takes two potential solutions and generates a crossover solution.

$$\boldsymbol{M} = \begin{pmatrix} f(K(1), K(1)) & \cdots & f(K(1), K(m)) \\ \vdots & \ddots & \vdots \\ f(K(m), K(1)) & \cdots & f(K(m), K(m)) \end{pmatrix} \tag{5.9}$$

The crossover function is random, and for every dimension of the solutions it can either take the value from the first solution, from the second solution, or generate a new value. In the case of continuous dimensions, such as θ or h_2, the crossover function can also yield the mean of the two solutions; this allows the solutions to approach a certain number with sufficient iterations without relying on it to be randomly generated. The probabilities for these may be heuristically chosen. It is necessary that the chance of the crossover function to choose a new random value (mutate) be large enough to continuously bring in new information into the system and allow it to keep improving without being so large that the newly generated population is less fit than the previous one.

For our implementation, separate crossover functions for continuous (θ and h_2) and discrete ($\boldsymbol{x_1}$) variables were used. These separate functions are given in Table I below. When new values are generated, they are drawn from the same prior distribution as in the initialization of the genetic algorithm.

Continuous (θ, h_2)	Discrete ($\boldsymbol{x_1}$)
$var_{new} = \begin{cases} var_1 & 30\% \\ var_2 & 30\% \\ \frac{var_1+var_2}{2} & 20\% \\ NewValue & 20\% \end{cases}$	$var_{new} = \begin{cases} var_1 & 45\% \\ var_2 & 45\% \\ NewValue, & 10\% \end{cases}$

Table 5.1: Crossover Function

For the discrete case, smaller probabilities of new values being generated were used due to the limited number of potential new values — there are only four possible values for a 4-QAM system for each variable x_1, so generating new values is less important, as the data likely exists somewhere in the population. On the other hand, for the continuous variables, high probabilities for combination and new value generation were used, ensuring that new values were constantly being added to the pool of solutions.

Following the generation of the $\boldsymbol{M}$ matrix, the genetic algorithm evaluates the cost function for every solution in $\boldsymbol{M}$. The m most fit solutions in the union of all solutions in $\boldsymbol{M}$ and the previous iteration's $\boldsymbol{K}$ vector are placed into the next iteration's $\boldsymbol{K}$ vector.

The genetic algorithm repeats the two steps of generating a new population to form a new $\boldsymbol{M}$ and pruning $\boldsymbol{M}$ to form $\boldsymbol{K}$ until convergence. Convergence can be gauged based on the variation of the solutions in $\boldsymbol{K}$; as an example, the algorithm could potentially be considered to have converged when the five most fit solutions in $\boldsymbol{K}$ all share the same decoding for $\boldsymbol{x_1}$. In such a scheme, any increase in the population size m should be followed by an increase in the probability of the crossover operation introducing new information to reduce the chance of premature convergence. To simplify hardware implementation, the number of iterations could be held constant, with the understanding that without ensuring convergence, there is potentially a greater chance of error during synchronization.

The genetic algorithm approach offers several advantages over other typical approaches. The bulk of the processing is required to calculate the fitness of the solutions in $\boldsymbol{M}$; the potentially large number of solutions in $\boldsymbol{M}$ suggests that the algorithm is

highly parallelizable, making it more practical for implementation than other typical stochastic optimization techniques.

5.2.2 Potential Improvements

This algorithm successfully manages to synchronize the interferer, but this is a computationally intensive process. An alternative method of synchronization could look not at the training data at the beginning of the interference frame, but rather at the data at the end. Due to our assumptions that the two frames are of equal length and the primary frame leads the interferer by at least twelve symbols, there are at least twelve interference-free data symbols on the tail end of the interferer. Blind synchronization techniques could potentially be used to provide good estimates for the channel gain and frequency offset. These estimates could then be passed to the synchronization algorithm as a prior distribution, significantly decreasing the computational cost of the joint synchronization/detection stage.

5.3 Joint Detection

Joint detection was performed by an algorithm based on a partial-joint detection algorithm described in [1]. Assuming U simultaneous interferers, all perfectly synchronized and with known discrete constellations, a near-ML Joint Minimum-Distance (JMD)

detector that decodes one signal is derived, expressed by

$$x_1 = \arg\min_{x_1} \left[\min_{x_2,\ldots,x_U} \left| y - h_1 x_1 - \sum_{u=2}^{U} h_u x_{u,m_u} \right|^2 \right] \tag{5.10}$$

where y is the received signal and x_U is the symbol from the constellation of transmitter U. As in our model, h_u corresponds to the channel gain from transmitter U to the receiver. Several modifications to this expression must be made to accurately reflect our implementation of this algorithm. Firstly, due to this paper being concerned with synchronizing only the first interferer, $U = 2$. The expression must furthermore be modified to return the associated best decoding for the interfering frame, $x_2(n)$. Taking into account the presence of the frequency offset and the synchronization-derived estimates for θ and h_2, the expression for the joint detector used is

$$x_1(n), x_2(n) = \arg\min_{x_1,x_2} \left[\left| y - h_1 x_1 - x_2 \hat{h_2} e^{jn\hat{\theta}} \right|^2 \right] \tag{5.11}$$

where $\hat{\theta}$ and $\hat{h_2}$ are the estimates of θ and h_2 returned by the synchronization algorithm. The primary weakness of this approach is its reliance on accurate estimates $\hat{h_2}$ and $\hat{\theta}$ and its inability to improve the estimates by observing errors during joint detection. This is especially problematic in the estimate $\hat{\theta}$, as even small errors in the frequency offset will compound with each new data symbol. The modification of the joint detection algorithm to continuously track the estimates $\hat{h_2}$ and $\hat{\theta}$ is left to future work.

5.4 Plain Interference Demodulation

Assuming that both frames are of the same length and the primary frame was the first to be received, then there will be a period during the detection during which the decoding of the primary frame is completed yet the interferer frame is still present. Assuming that the preamble of each frame contains information about the length of the frame, as in the 802.11 standard, the receiver has perfect knowledge of when this occurs, and so it can switch from the joint detection algorithm to a plain demodulation algorithm to decode x_2. For this section, the received signal is given by

$$y(n) = x_2 h_2 e^{jn\theta} \tag{5.12}$$

To solve n, a nearest-neighbor demodulator should be applied to the following symbol:

$$x_2(n) = \frac{y(n)}{\hat{h_2} e^{jn\hat{\theta}}} \tag{5.13}$$

Like the joint detection section, this algorithm will suffer from compounding error caused by a frequency offset estimation error during synchronization, and so to improve performance, a frequency offset tracking algorithm should be implemented.

Chapter 6

System Performance

In this section, the performance of the devised system relative to naive demodulation is evaluated by Monte Carlo simulation. The simulations assume a synchronized primary signal and an unsynchronized interfering frame, arriving at a random point during the duration of the primary frame. The first twelve symbols of the interferer are used for synchronization of the interferer, whereas the rest contain decodable data. Both frames are assumed to be modulated with 4-QAM, with a data length of 60 symbols. The frequency offset between the two frames is 2kHz, though a range of values was tested with similar results.

6.1 Interference Detection

The performance of the three devised interference detection schemes will here be examined. Two metrics of performance will be examined. The first of these is how well the algorithm is able to tell whether an interferer is present. This data will be presented

with receiver operating characteristics, a common way to present the performance of a binary classifier. The receiver operating characteristics plot the false positive rate versus the true positive rate for a certain receiver architecture. In practice, a certain receiver will operate at a certain point along this plot, at which the system designer decides the best trade-off between the false and true positive rates is.

The second metric has to do with how well the detector can estimate when in the stream the interferer arrives. If this is correctly estimated, than the system can continue to synchronize the interferer; if, however, an error occurs, than the synchronization will be done incorrectly, and the system may fail to resolve the collision. Note that the error rate here will be the chance of any error occuring at all, not the mean or mean-squared of the error.

6.1.1 No SNR or SIR Knowledge

Many synchronization schemes estimate the noise power, as the knowledge allows for more effective transmission strategies or even more effective decoding of the received data. Nevertheless, synchronization without estimating the noise power is possible. In such a system, the receiver would then not have the SNR available when trying to detect whether or not an interferer is present, and so other information must be used.

Figs. 6.1, 6.2 and 6.3 demonstrate the behavior of the detector at a real SIR of –5dB, 0dB and 5dB, respectively. Predictably, systems in which the interference is stronger — and SIR is thus smaller — have superior performance. At higher E_b/N_os, even systems at high SIRs are capable of correctly detecting the presence of interference.

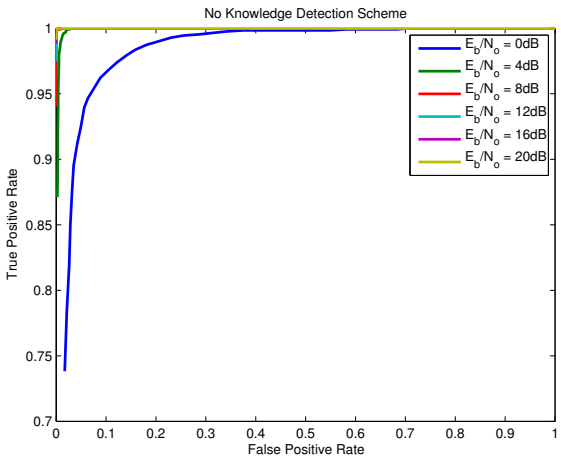

Figure 6.1: Blind Interference Detection for SIR = –5dB

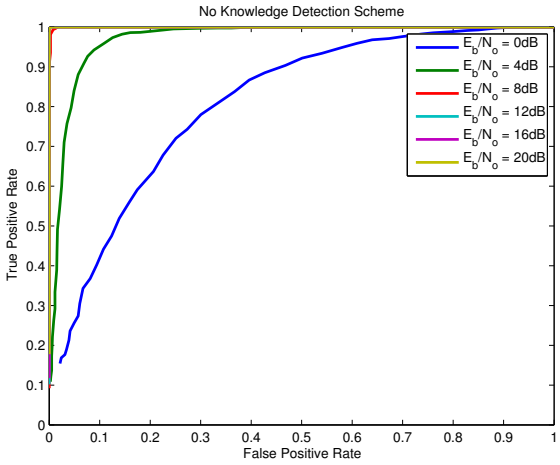

Figure 6.2: Blind Interference Detection for SIR = 0dB

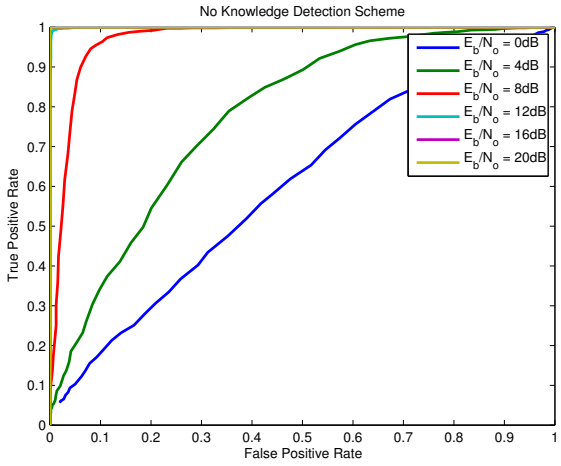

Figure 6.3: Blind Interference Detection for SIR = 5dB

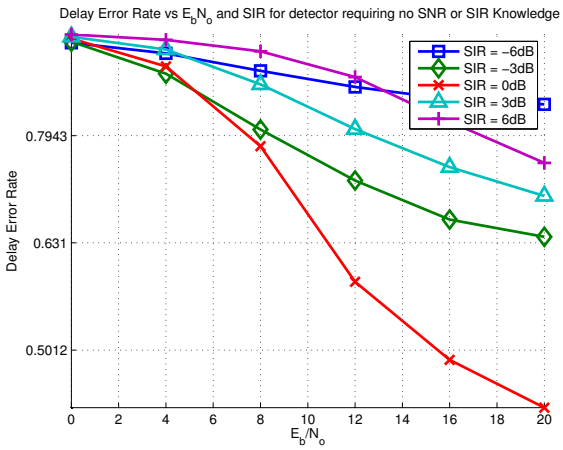

Figure 6.4: Blind Delay Estimation for SIR = 0dB

Fig. 6.4 shows that this algorithm performs poorly when it comes to correctly estimating the first symbol where the interferer is present. This level of performance is unacceptable in a modern communication system, at least if the rest of the system is the one discussed here. If the algorithm is not capable of accurately estimating the delay at which the interference begins, then the subsequent synchronization algorithm will fail, leading to a loss of both frames. Even at high SNRs and optimal SIRs, the delay estimation error rate is around 0.5, meaning that half of all collisions would not be detected.

Either a superior algorithm for this delay estimation must be developed using the given information, or more information must be provided to the receiver.

6.1.2 Full SIR and SNR Knowledge

There are systems in which the receiver may have knowledge of the SNR — acquired through typical synchronization mechanisms — and perhaps even prior expectation of any interferer SIR if there is a single expected interfering transmitter. In this case, the receiver has a lot of additional information to use to yield better detection results. Figs. 6.5, 6.6 and 6.7 show the receiver operating characteristics for SIRs of –5dB, 0dB and 5dB, respectively.

With full knowledge of both the SNR and the expected SIR, the binary detection of the interferer works very well. Apart from the cases where the E_b/N_o is 4 dB or less — which, even for a plain QPSK communication system, is very low — the detector is able to estimate the presence of interference very accurately. Unlike the algorithm without any SIR or SNR knowledge, the system is also capable at estimating the delay. For all SIRs except –6 dB, the system is able to estimate the delay correctly at least 90% of the time by $E_b/N_0 = 20$dB. In general, the algorithm tends to do better for SIRs near 0 dB. Unlike the previously shown system, then, this system performs sufficiently well that it could be worth implementing it to allow for joint decoding of two interfering frames.

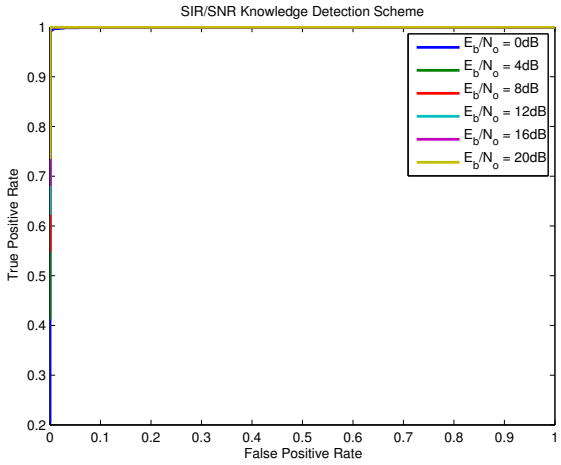

Figure 6.5: SNR and SIR-Aware Interference Detection for SIR = –5dB

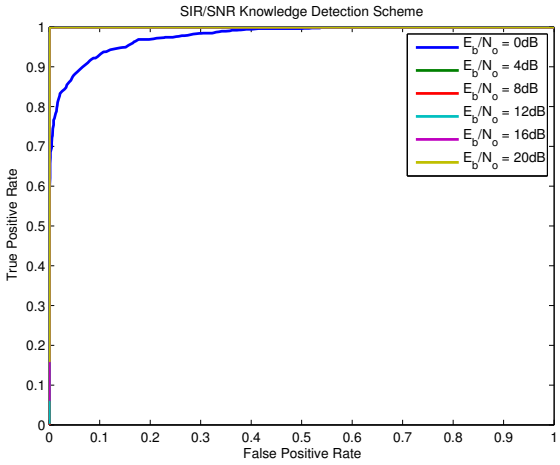

Figure 6.6: SNR and SIR-Aware Interference Detection for SIR = 0dB

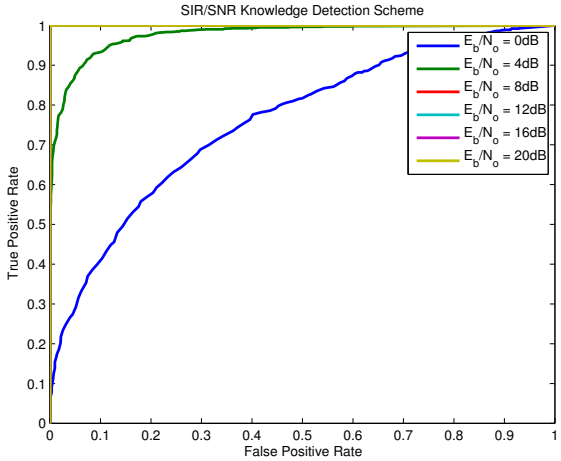

Figure 6.7: SNR and SIR-Aware Interference Detection for SIR = 5dB

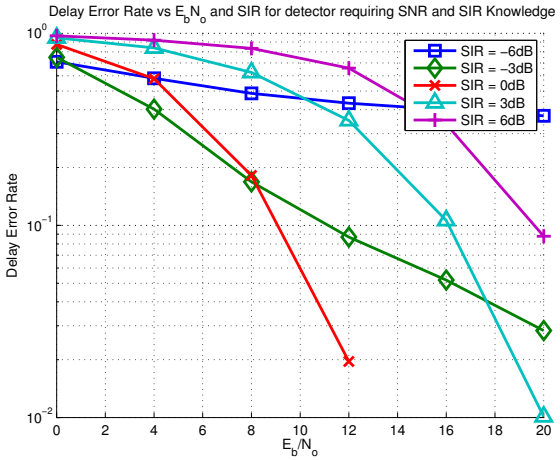

Figure 6.8: SNR and SIR-Aware Delay Estimation

6.1.3 Only SNR Knowledge

Perhaps most commonly, the receiver will have acquired knowledge of the SNR of the primary signal during synchronization, but there will be little prior knowledge about what the SIR will be during the collision. This may simply be because the power isn't predictable, or it may be caused by the inability of the receiver to track the information.

Figs. 6.9 and 6.10 show the binary detection rate of the receiver, and Figs. 6.11 and 6.12 show the delay-estimation performance of this algorithm. As a reminder, this algorithm works along similar lines as the algorithm that assumes knowledge of both

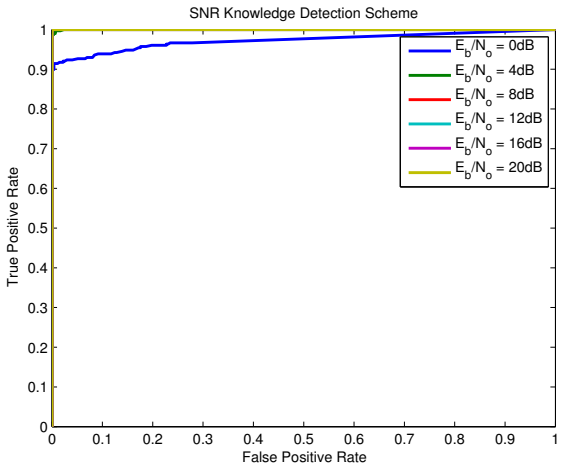

Figure 6.9: SNR-Aware Interference Detection for SIR = 0dB

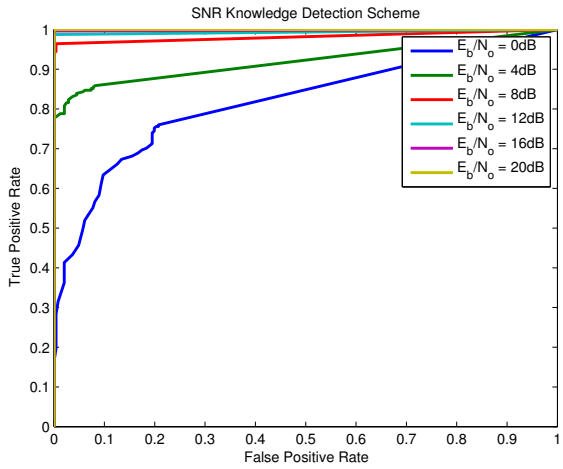

Figure 6.10: SNR-Aware Interference Detection for SIR = 5dB

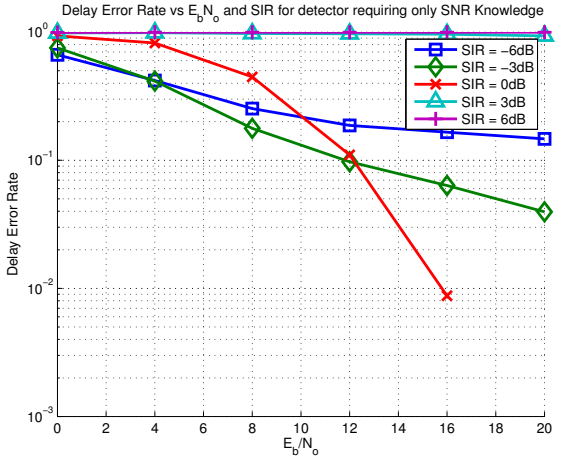

Figure 6.11: SNR-Aware Delay Estimation with SIR $\sim \mathcal{U}(-10dB, 10dB)$ prior

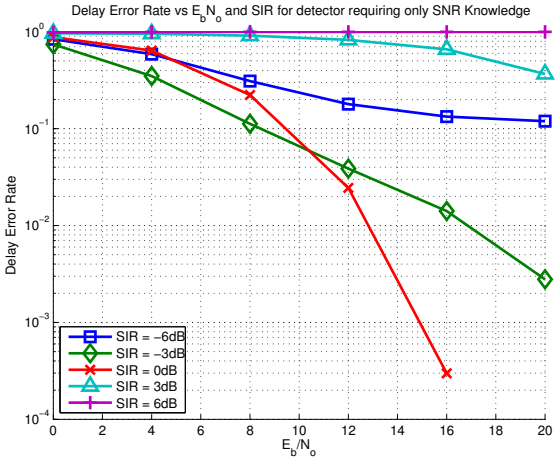

Figure 6.12: SNR-Aware Delay Estimation with SIR $\sim \mathcal{U}(-6dB, 6dB)$ prior

SNR and SIR, but it assumes a prior distribution on the SIR. For the ROC plots, this was chosen to be –6 dB to 6 dB.

The binary detection performance seems to be about where expected — superior to the algorithm with no information, but inferior to the algorithm with both SIR and SNR information. Detection rate of the interference at higher E_b/N_os is is acceptable for this kind of system, though it suffers at low E_b/N_0s and high SIRs. The delay estimation rate is poor at high SIRs, but markedly improves as the SIR is lowered. As the prior knowledge of the SIR is made more accurate, the delay estimation rate improves

significantly, particularly at low SIRs. Depending on how accurately the SIR of incoming frames is known in advance, this algorithm may be sufficiently accurate for practical implementation.

6.2 Joint Synchronization-Detection

The genetic synchronization-detection algorithm is the most complicated and difficult to mathematically model part of the system, and so demonstrating its effectiveness under a variety of conditions is particularly important. First, the effectiveness of the algorithm in a base case will be demonstrated. Then, various changes to the parameters of the algorithm will be made in an effort to quantify the effects of those changes, with everything else being held constant. Three metrics will be used to demonstrate the effectiveness of the algorithm: The BER rates of the primary frame, and the errors in the estimation of both the channel gain and frequency offset of the interferer.

The base case to be evaluated assumes that both the primary frame and the interferer use 4-QAM modulation. Both frames are assumed to use a training symbol length of 12 and a data length of 60 symbols. The training symbols associated with the primary frame are not simulated, as perfect synchronization of the primary frame is assumed. It is assumed that all the symbols have the same symbol period, and that the symbol period and receiver sampling period are the same. Perfect timing synchronization of both frames is assumed; though the exact time at which the interferer arrives it unknown, it is assumed to be quantized such that it sums ideally with the primary frame.

The interferer is assumed to arrive at the receiver at a time uniformly distributed

between the beginning of the data portion of the primary frame and twelve symbols away from the end of the primary frame. The first restriction ensures that that the primary frame is properly synchronized; the second ensures that there is always a primary frame to be decoded while the interferer is being synchronized. If the second restriction is not enforced, the consequence would be that the primary frame is no longer present during synchronization. The receiver would know this due to the frame length being stored in the preamble of a Wifi frame. Without an interferer present, the theoretical performance of the synchronization algorithm should greatly improve, though the code itself would need to be modified to accept this possibility.

The frequency offset to be estimated is 2kHz, while the channel gains are chosen to fit the simulated SIR with a uniform phase offset $\phi \in [0, 2\pi]$. The genetic algorithm is configured with $m = 60$, and with the algorithm performing 25 iterations before returning the estimates $\hat{\boldsymbol{x}_1}$, $\hat{\theta}$ and $\hat{h_2}$. Thus, unless explicitly stated otherwise, no convergence checking is performed.

Note that in the performed simulations, no forward-error correction or interleaving was performed. Implementing both of these would improve the system performance. Interleaving is important to implement due to the compounding of any error in the estimate θ — a consequence of this is that the symbols near the end of the collision would have a larger resulting phase offset, and thus a higher error rate. By essentially spreading these errors throughout the transmission and applying an error correction scheme, these errors could be mitigated even without implementing additional tracking features.

6.2.1 Base Results

Fig. 6.13 demonstrates the BER of the primary frame during joint synchronization-detection for the base case under a variety of SIRs and E_b/N_os. Some clear trends are observable. There is an error floor in the results. This is caused by the computational constraints of the algorithm — there is no ideal convergence criterion due to the presence of continuous variables, and so with a set number of iterations, the floor becomes iteration-limited rather than noise-limited. The error floor could be improved by dedicating more resources to the computation, such as by increasing m or the number of iterations of the algorithm. This is shown by comparing these results to those in the next section, where convergence detection is performed, leading to less accurate results. These error floors also improve as SIR increases.

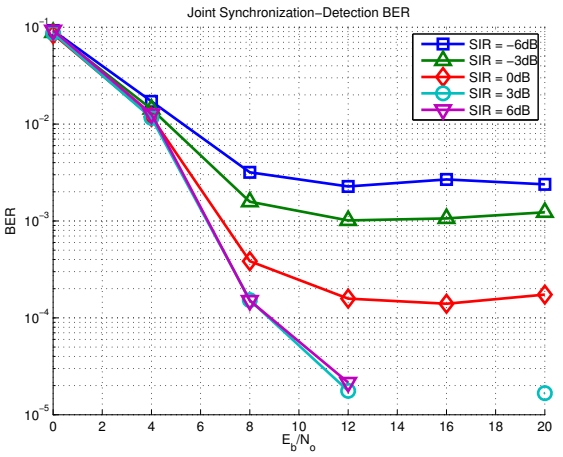

Figure 6.13: BER vs E_b/N_o and SIR for the base Joint Synchronization-Detection algorithm

The synchronization-stage mean-squared channel estimate error and mean frequency error are shown in Figs. 6.15 and 6.14, respectively. The synchronization algorithm dealing with a low-SIR system tends to provide superior estimates of the channel gain, but inferior estimates of the frequency offset. This in turn improves the BER for joint detection systems where there is a high SIR, as the frequency estimation error is the dominant source of error for the joint detection algorithm at high SNRs. Note that, as with the BER for the algorithm, there exists an error floor on the estimates.

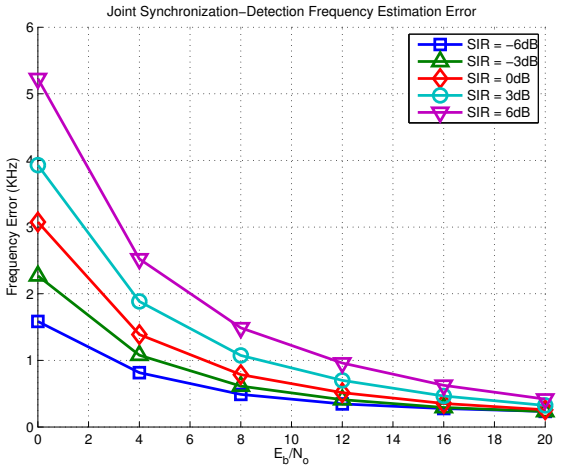

Figure 6.14: Mean Error in the Frequency Offset Estimation E_b/N_o and SIR for Base Algorithm

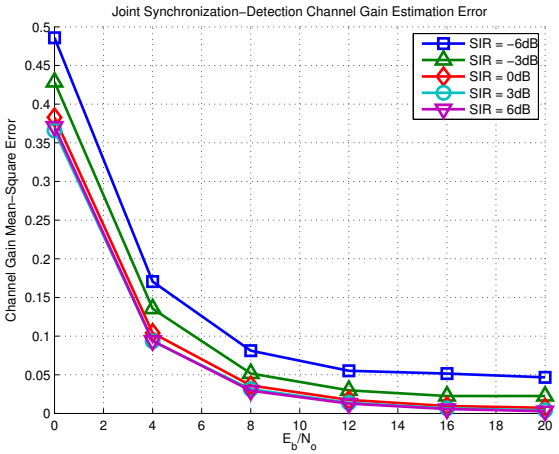

Figure 6.15: Mean-Squared Error in the Channel Gain Estimate vs E_b/N_o and SIR for Base Algorithm

6.2.2 Automatic Convergence

The algorithm could be modified to automatically converge. With the proper convergence criterion, this could help reduce the computational cost of the algorithm, though a specified number of iterations is likely simpler to implement in hardware. Without prior knowledge of the true values of $\boldsymbol{x_1}$, θ or h_2, convergence checking must

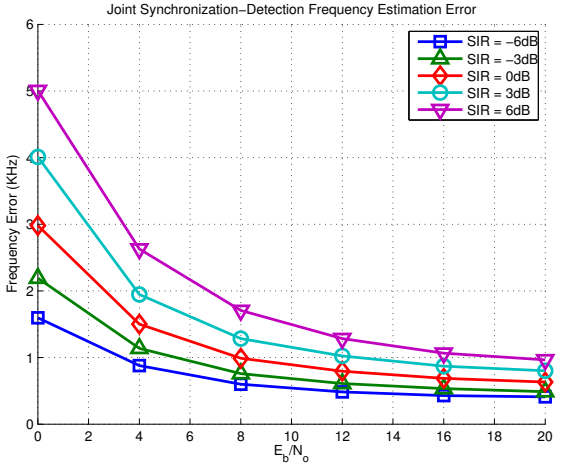

Figure 6.16: Mean Error in the Frequency Offset Estimation E_b/N_o and SIR for Convergence-Checking Algorithm

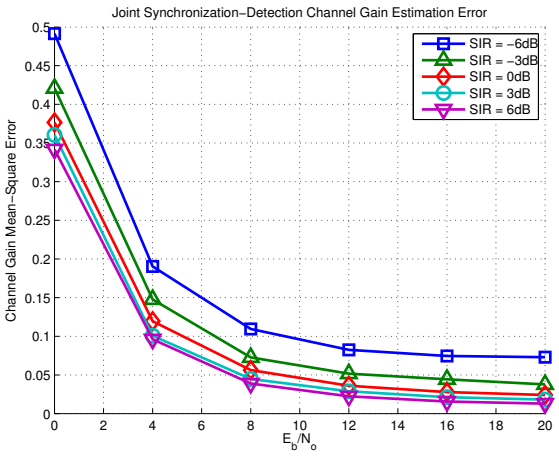

Figure 6.17: Mean-Squared Error in the Channel Gain Estimate vs E_b/N_o and SIR for Convergence-Checking Algorithm

be done by observing trends in the population of solutions. This could be done by looking at the variation of these variables over the population — the less variation there is in the solutions, the closer the algorithm is to convergence, as all solutions with radically different estimates were deemed to be poor and subsequently pruned. For this simulation, a simple convergence criterion was used; if the five most fit solutions from the vector $\boldsymbol{K}$ of $m = 60$ solutions all had identical estimates $\hat{\boldsymbol{x_1}}$, then the algorithm was judged to have converged. No more iterations were performed, and the data was passed to the joint detection algorithm.

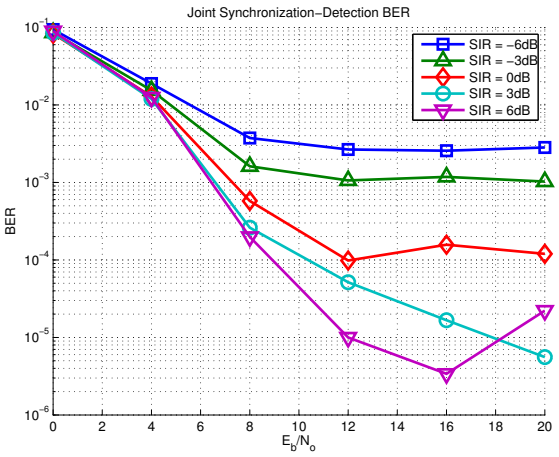

Figure 6.18: BER vs E_b/N_o and SIR for the Joint Synchronization-Detection Convergence-Checking Algorithm

The BER of $\boldsymbol{x_1}$ is shown here to be very similar to that with a set number of iterations. However, there are significant differences to be seen in the estimates for the channel gain and particularly the frequency offset. At an E_b/N_o of 20dB, for example, the average of the frequency offset error is about 1kHz for the automated convergence run but about 400Hz for 25 iterations. This strongly suggests that this convergence condition is insufficient if it is desired to achieve good joint decoding of both frames; the algorithm is converging without meeting the desired performance levels. It should thus be modified to take into account the variance of h_2 and θ.

6.2.3 Modifying Solution Population Size

The bulk of the processing cost of the genetic algorithm is the evaluation of the fitness of every solution. This cost scales with m^2, and so decreasing m could potentially lead to large computational complexity reductions. This prompted an experiment where m was reduced from 60 to 30; thus, the computational cost of the fitness-calculation portion of the algorithm was reduced by 75%.

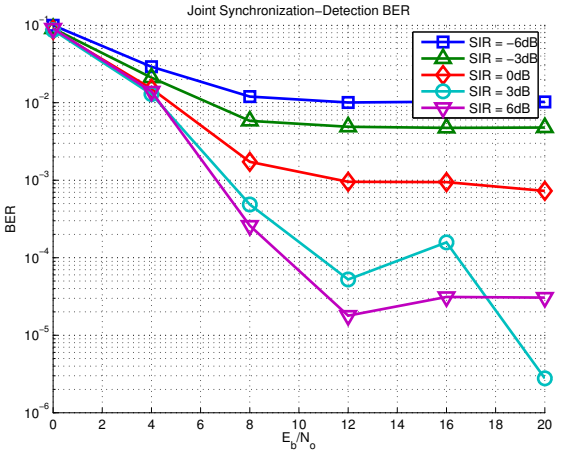

Figure 6.19: BER vs E_b/N_o and SIR for the Joint Synchronization-Detection for $m = 30$

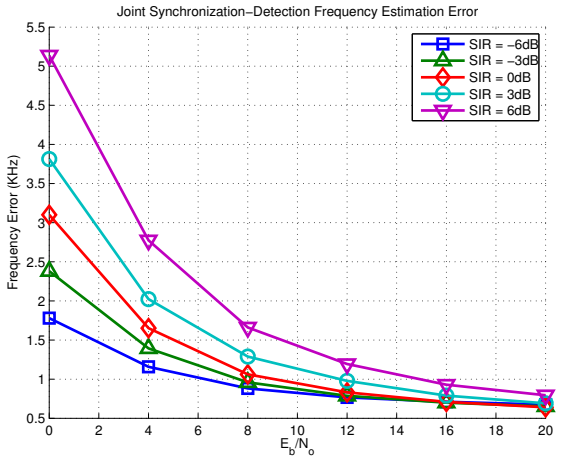

Figure 6.20: Mean Error in the Frequency Offset Estimation E_b/N_o and SIR for $m = 30$

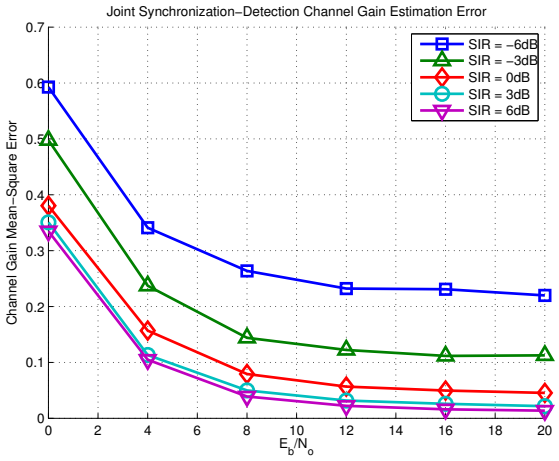

Figure 6.21: Mean-Squared Error in the Channel Gain Estimate vs E_b/N_o and SIR for $m = 30$

The effectiveness of the algorithm decreased drastically — especially at high SNRs — with m having been halved. The average over all SIRs of the frequency errors at E_b/N_o has increased from 400Hz to 750Hz. The channel gain error at an SIR of –6dB has increased from 0.05 to .3. Even the BER of $\boldsymbol{x_1}$ appears to have increased: At an SIR

of 6dB, the error floor appears to have risen from below 10^{-5} to 5×10^{-4}. This seems to have had a similar effect on the results as the automatic convergence.

6.2.4 BPSK Modulation

For this test, the modulation format was switched from 4-QAM (or QPSK) to BPSK, or Binary Phase-Shift Keying. The computational complexity of the genetic algorithm is greatly reduced, as the total solution space of $\boldsymbol{x_1}$ was reduced from $4^{12} = 1.67e7$ to $2^{12} = 4096$. This implies that the algorithm will successfully converge to the correct demodulation of $\boldsymbol{x_1}$ quicker, and thus have more iterations to spend refining the estimates $\hat{h_2}$ and $\hat{\theta}$. Furthermore, BPSK is more resistant to noise the QPSK, as the minimum distance between points on the constellation, d_{min}, is smaller.

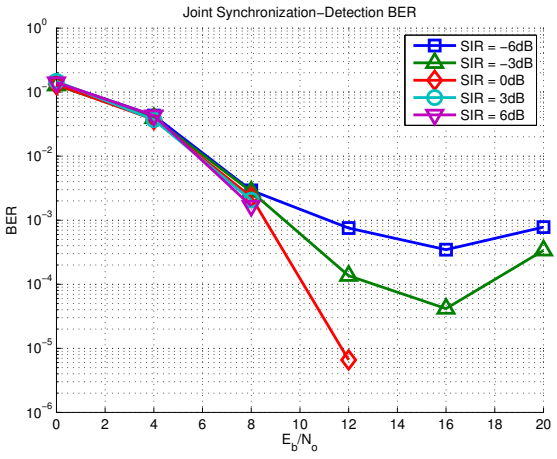

Figure 6.22: BER vs E_b/N_o and SIR for the Joint Synchronization-Detection for BPSK Modulation

The algorithm is particularly successful with this modulation scheme. This is an important result due to BPSK typically being used for control sequences, such as the preamble of a Wifi frame. In the event that the interferer arrives during the decoding of the preamble, the joint synchronization-detection algorithm will be more likely to correctly decode it. In the event that the preamble is decoded incorrectly, the remainder of the system would fail, as the receiver would have no information about the modulation

scheme and data length of the interferer.

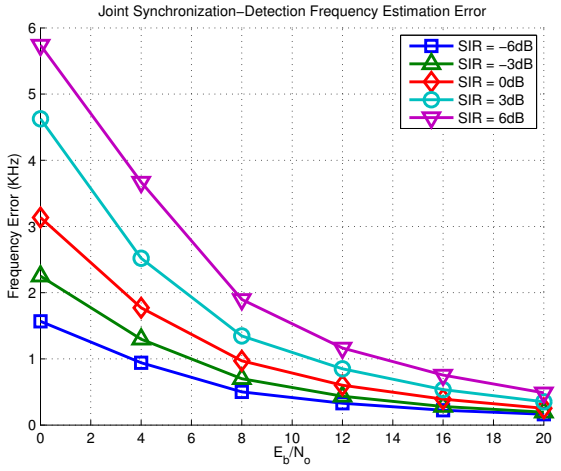

Figure 6.23: Mean Error in the Frequency Offset Estimation E_b/N_o and SIR for BPSK Modulation

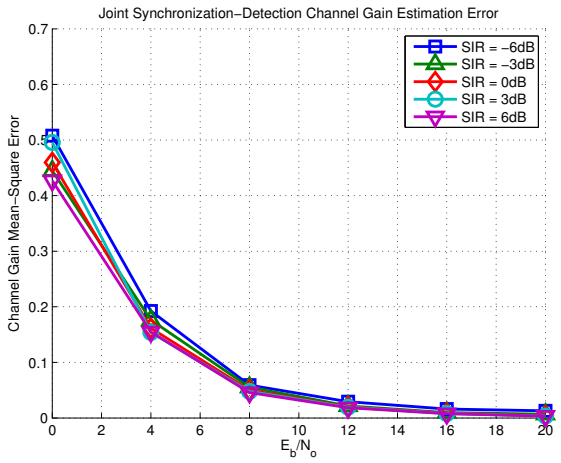

Figure 6.24: Mean-Squared Error in the Channel Gain Estimate vs E_b/N_o and SIR for BPSK Modulation

6.2.5 16-QAM Modulation

16-QAM Modulation, unlike BPSK, is more complex a modulation scheme than the base 4-QAM, and so the simulated results are expected to be inferior to those with 4-QAM. Compared to $1.67e7$ potential solutions for $\boldsymbol{x_1}$ in QPSK modulation, 16-QAM has $2.81e14$; thus, it is expected that more iterations are required to achieve similar computational performance. Furthermore, the distance between symbols is further compressed, reducing the efficacy of the modulation scheme at lower SNRs. Testing the performance of the algorithm at higher modulation

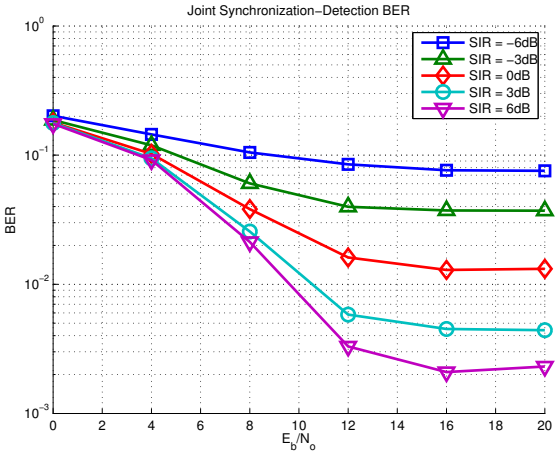

Figure 6.25: BER vs E_b/N_o and SIR for the Joint Synchronization-Detection for 16-QAM Modulation

schemes, such as 64-QAM, was considered, but due to the required computational complexity of the genetic algorithm to achieve useful results, rejected. The required increases in the number of iterations or m were too great to feasibly generate useful BER plots.

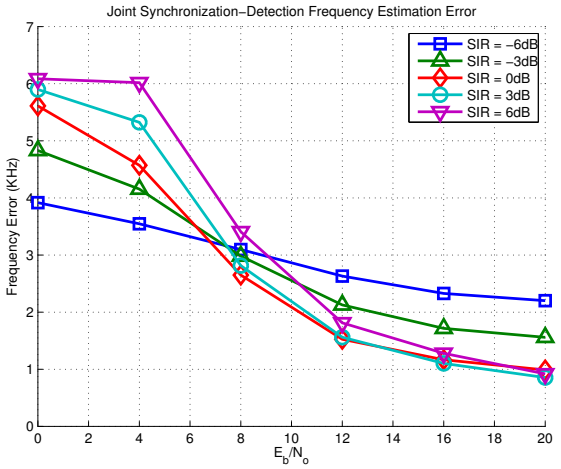

Figure 6.26: Mean Error in the Frequency Offset Estimation E_b/N_o and SIR for 16-QAM Modulation

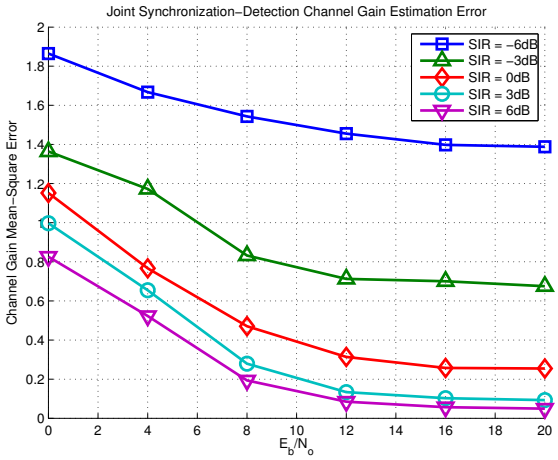

Figure 6.27: Mean-Squared Error in the Channel Gain Estimate vs E_b/N_o and SIR for 16-QAM Modulation

Fig. 6.25 shows that the BER has increased significantly due to the change in constellation, but despite that, the performance is still reasonable, at least at higher SIRs. Figs. 6.26 and 6.27 show that the channel and frequency estimates have also suffered, especially at low SIRs, but that results are still comparable at high SIRs and E_b/N_os, meaning that even with a huge increase in the size of the $\boldsymbol{x}_1$ being searched and without an increase of m, the algorithm can function reasonably well. The next section will explore the benefits of scaling m with changes in the constellation, weighing the increased complexity against the boost in performance.

6.2.6 16-QAM Modulation & $m = 100$

This section retains the 16-QAM modulation scheme from the last section, but attempts to increase m to from 60 to 100, yielding an increase from 3600 to 10000 solutions per iteration. The performance of the algorithm under these conditions is compared to that with a lower m.

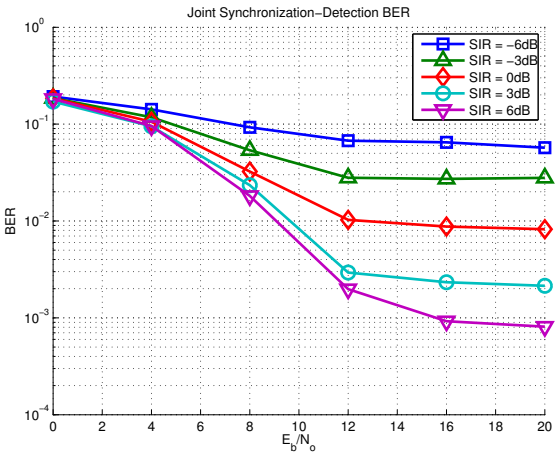

Figure 6.28: BER vs E_b/N_o and SIR for the Joint Synchronization-Detection for 16-QAM Modulation & $m = 100$

As before, the main weaknesses of the algorithm are in the low-SIR and low-SNR regime; where so little information is available about the interferer, that increasing the computational complexity returns little. Primarily, the increase yielded improvements where the algorithm already did well — high SIR and high SNR.

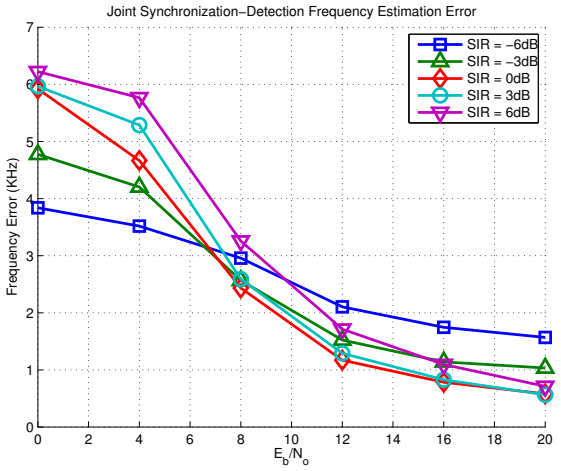

Figure 6.29: Mean Error in the Frequency Offset Estimation E_b/N_o and SIR for 16-QAM Modulation & $m = 100$

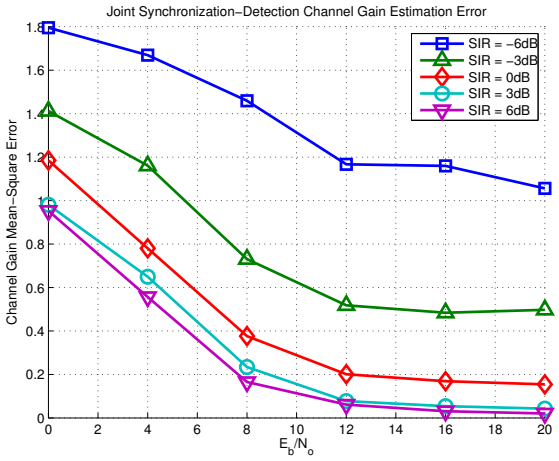

Figure 6.30: Mean-Squared Error in the Channel Gain Estimate vs E_b/N_o and SIR for 16-QAM Modulation & $m = 100$

The mean frequency estimation error for an SIR of 0 dB fell from 900Hz to 600Hz, while the BER for an SIR of 6dB fell from 2×10^{-2} to $.8 \times 10^{-3}$. These are significant improvements, but they show the pain of trading between performance and complexity — the improvements diminish as more computational power is provided to the algorithm.

6.3 Joint Detection Performance

Figs. 6.31, 6.32, 6.33 and 6.34 show the performance of the Joint Detection algorithm. The results computed using the estimates $\hat{\theta}$ and $\hat{h_2}$ provided by the synchronization algorithm are compared with the results given the true values of θ and h_2. They demonstrate the potential performance gains by implementing a frequency offset and channel gain tracking mechanism in the joint detection algorithm. It should also be noted that the performance of the joint detection algorithm given accurate values of θ and h_2 drastically improves as the SIR rises and falls from 0 dB; for an SIR of 5 dB, for example, the BER falls from 10^{-2} to 10^{-5} at $E_b/N_o = 20\ dB$.

Intuitively, the poor performance of the algorithm at SIRs close to 0 can be explained by noting that in the case that the SIRs are 0 and there is no phase offset between the two channel gains, the constellations as seen at the receiver are equivalent — thus, there is unavoidable ambiguity as to which transmitter sent each symbol. This ambiguity is extended to rotations of 90^o of the constellations (at least with 4-QAM), in which case, again, an ambiguity exists. Changing the power of the two interfering symbols with a constellation where all the powers are identical removes this ambiguity, yielding improved performance.

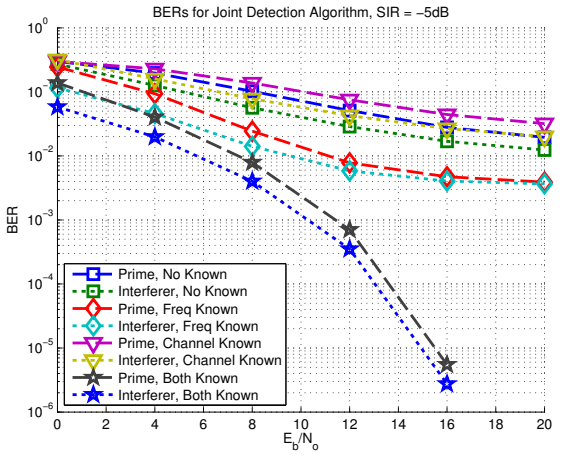

Figure 6.31: Joint Detection Performance for SIR = -5 dB

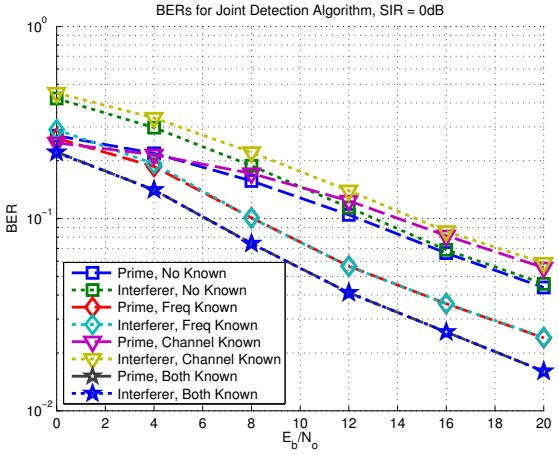

Figure 6.32: Joint Detection Performance for SIR = 0 dB

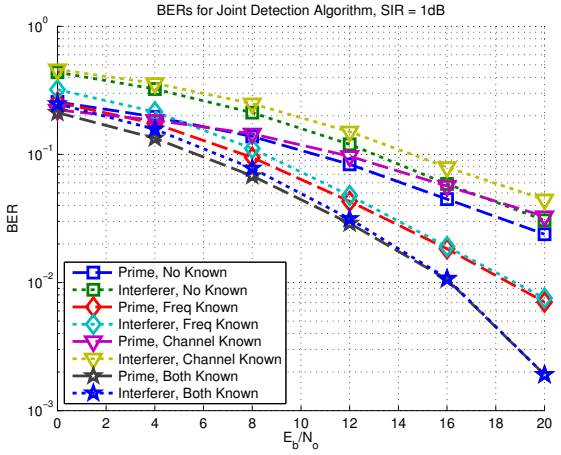

Figure 6.33: Joint Detection Performance for SIR = 1 dB

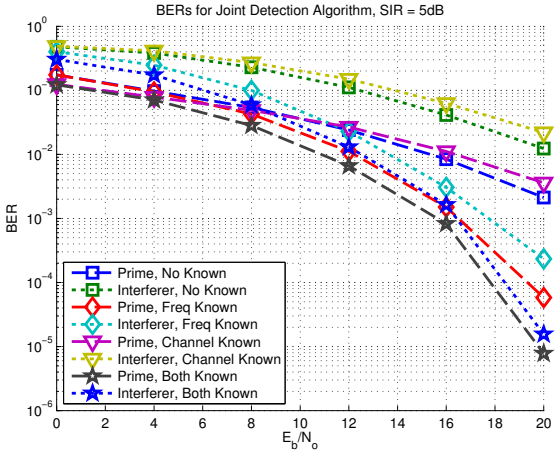

Figure 6.34: Joint Detection Performance for SIR = 5 dB

Some confusion may arise from differences in the performance of the algorithm at SIR = 5dB and SIR = –5dB; after all, with a symmetric joint detection algorithm, these should be identical, yet better BERs are achieved for SIR = –5dB. In effect, these two cases are identical, but accounting for the noise power is done with respect to different frames. For example, for an SIR of –5dB, the E_b/N_os of the two frames at the $E_b/N_o = 20\ dB$ point is 20 and 25; at the same point in the SIR = 5dB plot, these are 20 and 15.

Chapter 7

Conclusions and Further Work

This paper proposes a system that is capable of resolving collisions via joint detection. The system detects the presence of the interferer and performs channel and frequency synchronization. The synchronization data is used to perform joint detection, allowing the system to decode both the primary frame and the interfering frame.

Three separate interference detection algorithms are developed, two requiring both or neither SNR and SIR and the other requiring only SNR. Though all three algorithms are able to detect the presence of the signal to acceptable accuracy, only the one given both SNR and SIR is able to consistently estimate the beginning of the interferer. The blind detector is unable to achieve acceptable estimation rates for any SIR, while the detector with SNR knowledge is able to give reliable estimates for SIRs close to 0 dB.

The synchronization algorithm provides accurate estimates that could be improved by increasing the number of iterations. However, due to the noise, the performance may still not be sufficient to allow subsequent detection algorithms to reach their peak perfor-

mance. In particular, the frequency offset error propagates through the system, greatly increasing the error rate. This suggests that some form of tracking algorithm should be implemented, that would use the additional information from the joint detection to improve the estimates for the frequency offset and channel gain of the interferer.

A possible extension of the synchronization algorithm would attempt blind synchronization on the tail-end of the interferer, which is received interference-free but without knowledge of which symbols are being transmitted. This could help estimate the frequency offsets, channel gains and even help synchronize the interferer with the receiver in time. This information could then be combined with that gleaned by the synchronization algorithm to get refined estimates to help the joint detection algorithm approach its optimal performance.

Another possible extension of this project would be to enlarge the dimensionality, either by designing a system with multiple subcarriers or implementing this system for MIMO. This would potentially necessitate finding a more efficient synchronization algorithm. It could also prompt the development of a new joint detection algorithm, potentially inspired by the QRM-MLD or Sphere Decoder MIMO decoders, that could perform multidimensional underdetermined MIMO decoding more efficiently than the exhaustive search performed by the JMD algorithm.

Chapter 8

Appendix A: MATLAB Code

8.1 MIMO Decoder Comparison

The MIMO Decoder Comparison provides the code to generate Fig. 2.11. It's comprised of a general Plot Wrapper, a MIMO Encoder object, and several decoder objects/-functions. Note that the Sphere Decoder plotted was a native MATLAB implementation; no custom implementation was made. The algorithms will not necessarily work if significant changes are made; for example, the QRM-MLD implementation does not function beyond 2x2 MIMO. The decoders also assume that every available channel is used for data transmission, and that $N_T = N_R$; there is no support for utilizing the spatial streams for diversity.

8.1.1 MIMO Plotter

The MIMO Plotter acts as the testbench for all the MIMO algorithms. It sets up the modulator and demodulator objects, generates the data, passes it through the channel and through each decoder. It also handles the differing format of the outputs of the decoders — for example, the Minimum-Distance implementation returns the demodulated symbol numbers, whereas the Zero-Forcing algorithm returns the modulated symbols. To run quickly, reduce the variable numiter to something like '5'.

```
%% Andrew Apollonsky
clc;
clear all;
close all force;
%% Configuration
% Basic
datalen = 1000; % Length of each data stream
ebnos = 0:2:20; % E_b/N_os to test
```

```
numiter = 2500; % Number of iterations

% Channel
numTx = 2; % 2x2 MIMO
numRx = 2;
rS = 1;
maxDopp = 1e-1;
if numRx == numTx
    recCorrMat = eye(numRx);
    tranCorrMat = eye(numRx);
end
%% Object Creation
% Modulation
bpskmod = comm.BPSKModulator;
bpskdemod = comm.BPSKDemodulator;
qam4mod = comm.RectangularQAMModulator(4);
qam4demod = comm.RectangularQAMDemodulator(4);
qam4modb = comm.RectangularQAMModulator(4, 'BitInput', true);
qam4demodb = comm.RectangularQAMDemodulator(4, 'BitOutput', true);

modu = qam4mod; % Choose Modulator

if modu == qam4mod
    demod = qam4demod;
    modb = qam4modb;
elseif modu == bpskmod
    demod = bpskdemod;
    modb = bpskmod;
end

m = length(constellation(modu));

% MIMO
mimoenc = MIMOEnc;
mimodec1 = ZFMIMODec;
mimodec2 = MMSEMIMODec;
mimodec3 = MMSESICMIMODec;
mimodec4 = JMDMIMODec;
mimodec5 = comm.SphereDecoder('Constellation', constellation(modu), ...
    'BitTable', [0 0; 1 0; 0 1; 1 1], 'DecisionType', 'Hard');
% Replace the matrix [0 0; 1 0; 0 1; 1 1] above with [0 1].' for BPSK.

decs = {mimodec1, mimodec2, mimodec3, mimodec4, mimodec5};
needemod = [1 1 0 0 2 1 1]; % Tells script how the decoder outputs the data.

% Channel
chan1 = comm.MIMOChannel(...
    'SampleRate', rS, ...
    'MaximumDopplerShift', maxDopp, ...
    'PathGainsOutputPort', true, ...
    'TransmitCorrelationMatrix', tranCorrMat, ...
    'ReceiveCorrelationMatrix', recCorrMat);
chanresetiteration = 5;
chan2 = comm.AWGNChannel('NoiseMethod', 'Signal to noise ratio (SNR)');
```

```
wB = waitbar(0,'0%');

%% Simulation
hold all;
set(gca ,'yscale','log');
ylabel('BER');
xlabel('E_b/N_o');
grid on;
title('2x2 MIMO QPSK E_b/N_o vs BER');

tests = length(needemod);
prog = 0;
datadec2 = zeros(datalen*numTx, 1);
tic;
for reciter = 1:tests % Iterate over receivers
    for n = 1:length(ebnos) % Iterate over E_b/N_os
        ebno = ebnos(n);
        for p = 1:numiter % Iterate over iterations
            %% Data Generation
            data1 = randi([0 m-1], datalen * ...
                min(numTx, numRx), 1); % Generation
            datamod1 = step(modu, data1); % Modulation
            dataoenc1 = step(mimoenc, datamod1, numTx); % MIMO Encoding

            %% Channel Implementation
            if mod(p ,chanresetiteration) == 0
                reset(chan1);
            end
            [dataRay1, pathG1] = step(chan1, dataoenc1); % Channel
            chan2.SNR = ebno + 10*log10(log2(m));
            siz = size(dataRay1);
            chan2.SignalPower = sum(sum(abs(dataRay1.^2)))/(siz(1)*siz(2));
            rxSig1 = step(chan2, dataRay1); % AWGN Channel

            %% Decoding
            if reciter == 1
                datadec1 = step(decs{reciter}, rxSig1, squeeze(pathG1), numTx);
            elseif reciter == 2
                datadec1 = step(decs{reciter}, rxSig1, squeeze(pathG1), ...
                    var(rxSig1(:, 1) - dataRay1(:, 1)), numTx);
            elseif reciter == 3
                datadec1 = step(decs{reciter}, rxSig1, squeeze(pathG1), ...
                    var(rxSig1(:, 1) - dataRay1(:, 1)), modu, demod,...
                    numTx);
            elseif reciter == 4
                datadec1 = step(decs{reciter}, rxSig1, ...
                    squeeze(pathG1), modu, modb, numTx, numRx);
            elseif reciter == 5
                datadec1 = step(decs{reciter}, rxSig1, squeeze(pathG1));
            elseif reciter == 6
                datadec1 = qrm_mld(rxSig1, squeeze(pathG1), modu, 2);
            elseif reciter == 7
                datadec1 = qrm_mld(rxSig1, squeeze(pathG1), modu, 4);
            end
```

```
            %% Serialize, convert to proper format
            if needemod(reciter) == 1
                datadec3 = reshape(datadec1.', [], 1);
                datadec4 = step(demod, datadec3);
            elseif needemod(reciter) == 0
                datadec4 = reshape(datadec1.', [], 1);
            elseif needemod(reciter) == 2
                datadec2 = reshape(datadec1, log2(m), []).';
                datadec3 = bi2de(datadec2, log2(m));
                datadec4 = zeros(length(datadec1), 1);
                datadec4(1:numRx:length(datadec2)) = ...
                    datadec3(1:length(datadec2)/2);
                datadec4(2:numRx:length(datadec2)) = ...
                    datadec3(length(datadec2)/2+1:length(datadec2));
            end

            %% Find BER
            ber(n, p) = mean(reshape(de2bi(data1, log2(m)), [], 1) ...
                ~= reshape(de2bi((datadec4+0), log2(m)), [], 1));
        end
        prog = prog + 1/(length(ebnos) * tests);
        waitbar(prog, wB, strcat(int2str(round(prog*100)), '%'));
    end
    plot(ebnos, mean(ber, 2), 'LineWidth', 2);
end
toc
close(wB);

legend('ZF', 'MMSE', 'MMSE-SIC', 'Minimum-Distance', 'Sphere Decoder',...
    'QRM-MLD, M = 2', 'QRM-MLD, M = 4');
set(gcf, 'Color', 'w');
export_fig MIMOPlots.eps
```

8.1.2 MIMO Encoder

The MIMO Encoder takes the incoming data stream and separates it among the desired number of MIMO channels.

```
classdef MIMOEnc < matlab.System
    methods (Access=protected)
        function out = stepImpl(~, data, numTx)
            out = zeros(length(data)/numTx, numTx);
            for k = 1:numTx
                out(:, k) = data(k:numTx:end);
            end
        end

        function numIn = getNumInputsImpl(~)
           numIn = 2;
        end
```

```
        function numOut = getNumOutputsImpl(~)
           numOut = 1;
        end
    end
end
```

8.1.3 Zero-Forcing

Implementation of the linear Zero-Forcing MIMO Decoder.

```
classdef ZFMIMODec < matlab.System
    methods (Access=protected)
        function out = stepImpl(~, data, PG, numTx)
            out = zeros(length(data), numTx);
            W = zeros(numTx, numTx, length(data));
            for t = 1:length(data)
                k = squeeze(PG(t, :, :));
                W(:, :, t) = (k'*k)\k';
                out(t, :) = data(t, :)*W(:, :, t);
            end
        end

        function numIn = getNumInputsImpl(~)
           numIn = 3;
        end

        function numOut = getNumOutputsImpl(~)
           numOut = 1;
        end
    end
end
```

8.1.4 MMSE

Implementation of the linear MMSE MIMO Decoder.

```
classdef MMSEMIMODec < matlab.System
    methods (Access=protected)
        function out = stepImpl(~, data, PG, N0, numTx)
            out = zeros(length(data), numTx);
            W = zeros(numTx, numTx, length(data));
            for t = 1:length(data)
                k = squeeze(PG(t, :, :));
                W(:, :, t) = (k'*k + eye(numTx)*N0)\k';
                out(t, :) = data(t, :)*W(:, :, t);
            end
        end
```

```
        function numIn = getNumInputsImpl(~)
           numIn = 4;
        end

        function numOut = getNumOutputsImpl(~)
           numOut = 1;
        end
    end
end
```

8.1.5 MMSE-SIC

Implementation of the MMSE-SIC MIMO Decoder.

```
classdef MMSESICMIMODec < matlab.System
    methods (Access=protected)
        function out = stepImpl(~, data, PG, N0, mod, demod, numTx)
            out = zeros(length(data), numTx);
            pow = sum(PG.^2, 3);
            [~, powind] = sort(pow, 2, 'descend');
            for t = 1:length(data)
                PGT = squeeze(PG(t, :, :)).';
                data2 = data(t, :);
                for k = 1:numTx
                    W = (PGT'*PGT + eye(numTx)*N0)\PGT';
                    temp = W*data2.';
                    xhat = temp(powind(t, k));
                    out(t, powind(t, k)) = step(demod, xhat);
                    mods = step(mod, out(t, powind(t, k)));
                    data2 = data2 - PGT(:, powind(t, k)).' * mods;
                end
            end
        end

        function numIn = getNumInputsImpl(~)
           numIn = 6;
        end

        function numOut = getNumOutputsImpl(~)
           numOut = 1;
        end
    end
end
```

8.1.6 Minimum-Distance

Implementation of the Minimum-Distance MIMO decoder. Supports up to 8x8 MIMO.

```
classdef JMDMIMODec < matlab.System
    methods (Access=protected)
        function out = stepImpl(~, data, PG, mod, modb, numTx, numRx)
            % Matrix preallocation
            m = length(constellation(mod));
            out = zeros(length(data), numTx);
            mvec = [ones(numTx, 1)*m; ones(8-numTx, 1)];
            minimat1 = zeros(length(data), numTx, numRx, m);
            minimat2 = zeros(length(data), numRx, mvec(1), mvec(2), ...
                mvec(3), mvec(4), mvec(5), mvec(6), mvec(7), mvec(8));
            grid1 = ones(length(data), mvec(1), mvec(2), mvec(3), mvec(4),...
                mvec(5), mvec(6), mvec(7), mvec(8), numTx);

            % Generation of values for minimum-distance detector
            for p = 1:numTx
                for n = 0:m-1
                    aa = step(mod, n);
                    for k = 1:numRx
                        minimat1(:, p, k, n+1) = PG(:, p, k) * aa;
                    end
                end
            end

            % Grid for repmatting later.
            gridthing(1, :) = [1 2 3 4 5 6 7 8 9];
            for k = 2:8
                gridthing(k, :) = gridthing(k-1, :);
                gridthing(k, [k+1, k]) = gridthing(k, [k, k+1]);
            end

            % Generating values that will be subtracted for JMD receiver.
            for k = 1:numRx
                for p = 1:numTx
                    grid1(:, :, :, :, :, :, :, :, :, p) = repmat(permute(...
                        squeeze(minimat1(:, p, k, :)), gridthing(p, :)), ...
                        [1 ones(1, p-1)*m 1 ones(1, numTx-p)*m]);
                end
                minimat2(:, k, :, :, :, :, :, :, :, :) = permute(sum(grid1,...
                    10), [1 10 2 3 4 5 6 7 8 9]);
            end


            % Calculating Maximat, which stores the differences between the
            % actual data and would-be data for every single combination.
            maximat = permute(squeeze(sum(abs(squeeze(repmat(data, [1, 1, ...
                mvec(1), mvec(2), mvec(3), mvec(4), mvec(5), mvec(6), ...
                mvec(7), mvec(8)])) - squeeze(minimat2)), 2)), [2 3 4 5 6 7 8 9 1]);

            % Using the values to find the decoded signal.
            for t = 1:length(data)
                [~, minind] = ...
                    min(reshape(maximat( :, :, :, :, :, :, :, :, t), [], 1));
                siz = size(maximat(:, :, :, :, :, :, :, :, t));
                [a, b, c, d, e, f, g, h] = ind2sub(siz, minind);
```

```
                k = [a b c d e f g h];
                out(t, :) = k(1:numTx)-1;
            end
        end

        function numIn = getNumInputsImpl(~)
           numIn = 6;
        end

        function numOut = getNumOutputsImpl(~)
           numOut = 1;
        end
    end
end
```

8.1.7 QRM-MLD

Implementation of QRM-MLD MIMO Decoder. Supports up to 2x2 MIMO.

```
function [datout] = qrm_mld(data, PG, mod, M)
    const = constellation(mod);
    datout = zeros(length(data), 2);
    for t = 1:length(data) % Loop over time
        [Q, R] = qr(squeeze(PG(t, :, :)).'); % QR Decomposition
        yhat = Q'*(data(t, :).'); % Generate Q*y
        dist = zeros(length(const), 1); % Distance Preallocation
        for a = 1:length(const)
            dist(a) = abs(yhat(2) - R(2, 2)*const(a)); % Calculate distance
        end
        [~, I] = sort(dist, 'ascend'); %Find best solutions
        x2s = const(I(1:min(length(const), M))); % Grab best solutions
        dist = zeros(length(x2s), length(const)); % Distance Preallocation
        for a = 1:length(x2s) % Loop over last iteration
            for b = 1:length(const) % Loop over current possibilities
                dist(a, b) = (yhat(2) - R(2, 2)*x2s(a))^2 + ...
                    (yhat(1) - R(1, 1)*const(b) - R(1, 2)...
                    *x2s(a))^2; % Calculate distance
            end
        end
        [~, minind] = min(reshape(dist, [], 1)); % Find best index
        [a, b] = ind2sub(size(dist), minind); % Find corresponding symbols
        datout(t, :) = [const(b) x2s(a)]; % Output them
    end
end
```

8.2 Interference System

This section contains the MATLAB code that simulates the behavior of the derived system.

8.2.1 System Plotter

The system plotter simulates the performance of the entire system over a specified list of SIRs and E_b/N_os. The system contains code to generate the plots that show synchronization performance in terms of frequency estimation, channel estimation and simultaneous decoding BER. The system also contains code to generate plots comparing the performance of the joint detection algorithm with the estimated parameters versus with ideal knowledge of the system parameters.

```
%% Andrew Apollonsky
close all force;
clc;
clear all;
%% Simulation Parameters
len1 = 12; % Training length
len2 = 60; % Frame length (discounting training)
m1 = 4;
m2 = 4;

ebnos = 0:4:20; % E_b/N_os to simulate
sirs = [-6:3:6]; % SIRs to simulate
numiter = 15; % Number of iterations per E_b/N_o and SIR

ts = 4e-6; % Sample Time
freq = 2.4; % Simulated Frequency

maxDopp = 1;
rotfacreal = .05;
realfreqoffset = rotfacreal /(2*pi) / ts / 1e3;%Real frequency offset, in KHz

%% Object Initialization
bpskmod = comm.BPSKModulator(0);
bpskdemod = comm.BPSKDemodulator(0);
qam4mod = comm.RectangularQAMModulator(4);
qam4demod = comm.RectangularQAMDemodulator(4);
qam16mod = comm.RectangularQAMModulator(16);
qam16demod = comm.RectangularQAMDemodulator(16);
qam64mod = comm.RectangularQAMModulator(64);
qam64demod = comm.RectangularQAMDemodulator(64);
qam256mod = comm.RectangularQAMModulator(256);
qam256demod = comm.RectangularQAMDemodulator(256);

if m1 == 2
    mod1 = bpskmod;
    demod1 = bpskdemod;
elseif m1 == 4
    mod1 = qam4mod;
    demod1 = qam4demod;
elseif m1 == 16
    mod1 = qam16mod;
    demod1 = qam16demod;
end
```

```
if m2 == 2
    mod2 = bpskmod;
    demod2 = bpskdemod;
elseif m2 == 4
    mod2 = qam4mod;
    demod2 = qam4demod;
elseif m2 == 16
    mod2 = qam16mod;
    demod2 = qam16demod;
end

g1 = 1; % Gain of primary signal


bersgen = nan(length(sirs), length(ebnos), numiter);
berstand = nan(length(sirs), length(ebnos), numiter);
pgsguess = nan(length(sirs), length(ebnos), numiter);
rotfacguess = nan(length(sirs), length(ebnos), numiter);
pgsreal = nan(length(sirs), length(ebnos), numiter);
numiters = nan(length(sirs), length(ebnos), numiter);
delayerr = nan(length(sirs), length(ebnos), numiter);
jmdber11 = nan(length(sirs), length(ebnos), numiter);
jmdber12 = nan(length(sirs), length(ebnos), numiter);
jmdber21 = nan(length(sirs), length(ebnos), numiter);
jmdber22 = nan(length(sirs), length(ebnos), numiter);
jmdber31 = nan(length(sirs), length(ebnos), numiter);
jmdber32 = nan(length(sirs), length(ebnos), numiter);
jmdber41 = nan(length(sirs), length(ebnos), numiter);
jmdber42 = nan(length(sirs), length(ebnos), numiter);

out3ber = nan(length(sirs), length(ebnos), numiter);
sigpresentest = nan(length(sirs), length(ebnos), numiter);
sigpresentreal = nan(length(sirs), length(ebnos), numiter);

wB = waitbar(0,'0%');
prog = 0;
%% Calculations
[evm1, evm2, var1, var2] = errcalc(sirs, ebnos + 10*log10(log2(m1)), ...
    1e5, mod1, demod1, mod1);

[evm11, evm22, var11, var22] = errcalc(-10:2:10, ebnos + ...
    10*log10(log2(m1)), 1e5, mod1, demod1, mod1);
tic;
for siriter = 1:length(sirs)
    sir2 = sirs(siriter) - 10*log10(log2(m1)) + 10*log10(log2(m2));
    g2 = 10^(-sir2/20);
    for ebnoiter = 1:length(ebnos)
        SNR = ebnos(ebnoiter) + 10*log10(log2(m1));
        for iter = 1:numiter
            sig2delay = randi([1 len2-len1 - 1]); % Generate random delay
            totlen = sig2delay + len1 + len2; % Compute total simulation length

            % Generate data.
```

```
x1 = [randi([0 m1-1], len2, 1); zeros(sig2delay + len1, 1)];
x2 = [zeros(sig2delay, 1); randi([0 1], len1, 1);...
    randi([0 m2-1], len2, 1)];

% Modulated data. 0 where no signal exists.
y1 = [step(mod1, x1(1:len2)); zeros(sig2delay + len1, 1)];
y2 = [zeros(sig2delay, 1); ...
    step(bpskmod, x2(sig2delay+1:sig2delay+len1)); ...
    step(mod2, x2(sig2delay+len1+1:end))];

% Real path gains with specified power and random phase
% generated
pathG1 = [ones(len2, 1) * exp(rand(1) * 2*pi*1i) * g1; ...
    zeros(len1 + sig2delay, 1)];
pathG2 = [zeros(sig2delay, 1); ones(len1 + len2, 1) ...
    * exp(rand(1) * 2*pi*1i) * g2];

% Apply frequency-offset-induced rotation in path gain
for t = sig2delay+1:totlen
    pathG2(t) = pathG2(t) * ...
        exp((t-sig2delay-1) * 1i * rotfacreal);
end

% Apply path gains to data
rx1 = pathG1 .* y1;
rx2 = pathG2 .* y2;

% Add AWGN
rx1 = awgn(rx1, SNR, mean(abs(y1(1:len2)).^2));

% Sum signals. 50% chance of no interferer, no summation.
if rand(1) > .5
    rxSig = rx1 + rx2;
    sigpresreal = 1;
else
    rxSig = rx1;
    sigpresreal = 0;
end

% Find if signal present, and if so, find the delay
% Uses full knowledge of SIR and SNR
[delayest, sigpres] = delayfind(rxSig, mod1, demod1, pathG1, ...
    len2, evm1(siriter, ebnoiter), evm2(siriter, ebnoiter), 0);

% Uses no knowledge of SIR or SNR -- Doesn't work very well
%   [delayest, sigpres] = delayfind2(rxSig, mod1,...
%       demod1, pathG1, len2, .02); % Threshold may need adjusting

% Uses knowledge of SNR and assumption that -10dB<SIR<10dB
%   [delayest, sigpres] = delayfind(rxSig, mod1, demod1, pathG1, ...
%       len2, mean(evm11(:, ebnoiter)), mean(evm22(:, ebnoiter)), 0);

if sigpres == 1 && sigpresreal == 1
    if sig2delay == delayest
```

```
% Apply Synchronization Algorithm
[out1gen, wts2, iters, pgbase, rotfac] = ...
    gen_al(rxSig(delayest + 1:delayest + len1), ...
    y2(delayest+1:delayest + len1), mod1, ...
    pathG1(delayest+1:delayest + len1));

% Estimate future path gain for signal 2
% Uses estimated frequency and channel
pg2 = pgbase * exp(1i * rotfac * ...
    [0:len1 + len2-1]).';
% Uses real frequency information
pg22 = pgbase * exp(1i * rotfacreal * ...
    [0:len1 + len2-1]).';
% Uses real channel information
pg23 = pathG2(sig2delay+1) * ...
    exp(1i * rotfac * [0:len1 + len2-1]).';
% Uses real frequency and channel information (e.g.
% ideal case)
pg24 = pathG2(sig2delay+1) * ...
    exp(1i * rotfacreal * [0:len1 + len2-1]).';

% Joint Detection
[out1, out2] = ...% Using Synchronization Data
    JMD(rxSig(delayest+len1+1:len2), mod1, mod2, ...
    pathG1(delayest+len1+1:len2),...
    pg2(len1+1:len2-delayest));

[out12, out22] = ... % Using Real Frequency
    JMD(rxSig(delayest+len1+1:len2), mod1, mod2, ...
    pathG1(delayest+len1+1:len2),...
    pg22(len1+1:len2-delayest));

[out13, out23] = ... % Using Real Channel
    JMD(rxSig(delayest+len1+1:len2), mod1, mod2, ...
    pathG1(delayest+len1+1:len2),...
    pg23(len1+1:len2-delayest));

[out14, out24] = ... % Using Real Chan & Freq
    JMD(rxSig(delayest+len1+1:len2), mod1, mod2, ...
    pathG1(delayest+len1+1:len2),...
    pg24(len1+1:len2-delayest));

% Finish Detecting - The stage after Joint Detection
out3demod = step(demod2, rxSig(len2+1:end)...
    ./ pg2(len2-delayest+1:end)); % Using Synch. Data
out3demod2 = step(demod2, rxSig(len2+1:end)...
    ./ pg22(len2-delayest+1:end)); % Using Real Freq
out3demod3 = step(demod2, rxSig(len2+1:end)...
    ./ pg23(len2-delayest+1:end)); % Using Real Chan
out3demod4 = step(demod2, rxSig(len2+1:end)...
    ./ pg24(len2-delayest+1:end)); % Using Real Both

% Demodulate synchronization things
out1demodgen = step(demod1, out1gen);
```

```
out1demodstand = step(demod1, ...
    rxSig(delayest+1:delayest+len1)...
    ./ pathG1(delayest+1:delayest+len1));

%% Binary Conversion
% Genetic Algorithm
out1demodgenbin = ...
    reshape(de2bi(out1demodgen, log2(m1)), [], 1);
out1demodstandbin = ...
    reshape(de2bi(out1demodstand, log2(m1)), [], 1);

% Tested JMD Performance
out1bin = reshape(de2bi(out1, log2(m1)), [], 1);
out2bin = reshape(de2bi(out2, log2(m2)), [], 1);

% Prior knowledge of freq
out12bin = reshape(de2bi(out12, log2(m1)), [], 1);
out22bin = reshape(de2bi(out22, log2(m2)), [], 1);

% Prior knowledge of channel
out13bin = reshape(de2bi(out13, log2(m1)), [], 1);
out23bin = reshape(de2bi(out23, log2(m2)), [], 1);

% Prior knowledge of both
out14bin = reshape(de2bi(out14, log2(m1)), [], 1);
out24bin = reshape(de2bi(out24, log2(m2)), [], 1);

% Single—Detection of frame 2
out3bin = reshape(de2bi(out3demod, log2(m2)), [], 1);
out32bin = reshape(de2bi(out3demod2, log2(m2)), [], 1);
out33bin = reshape(de2bi(out3demod3, log2(m2)), [], 1);
out34bin = reshape(de2bi(out3demod4, log2(m2)), [], 1);

% Real values
realgenbin = reshape(de2bi(...
    x1(sig2delay+1:sig2delay+len1), log2(m1)), [], 1);
realjmdx1b = reshape(de2bi(...
    x1(sig2delay + len1 + 1:len2), log2(m1)), [], 1);
realjmdx2b = reshape(de2bi(...
    x2(sig2delay + len1 + 1:len2), log2(m2)), [], 1);
realnormb = reshape(de2bi(...
    x2(len2+1:end), log2(m2)), [], 1);

%% Find BERs
% Genetic Algorithm
synchgenber = sum(out1demodgenbin ...
    ~= realgenbin)/length(realgenbin);

% Interference ignorant demodulator
synchstandber = sum(out1demodstandbin ...
    ~= realgenbin)/length(realgenbin);

% JMD — No Ideal Synchronization
out1ber1 = ...
```

```
                    sum(out1bin ~= realjmdx1b)/length(realjmdx1b);
                out2ber1 = ...
                    sum(out2bin ~= realjmdx2b)/length(realjmdx2b);

                % JMD - Prior Frequency Knowledge
                out1ber2 = ...
                    sum(out12bin ~= realjmdx1b)/length(realjmdx1b);
                out2ber2 = ...
                    sum(out22bin ~= realjmdx2b)/length(realjmdx2b);

                % JMD - Prior Channel Knowledge
                out1ber3 = ...
                    sum(out13bin ~= realjmdx1b)/length(realjmdx1b);
                out2ber3 = ...
                    sum(out23bin ~= realjmdx2b)/length(realjmdx2b);

                % JMD - Prior Channel & Frequency Knowledge
                out1ber4 = ...
                    sum(out14bin ~= realjmdx1b)/length(realjmdx1b);
                out2ber4 = ...
                    sum(out24bin ~= realjmdx2b)/length(realjmdx2b);

                % Single Detection - No Ideal Synchronization
                out3ber1 = sum(out3bin ~= realnormb)/length(out3bin);
            end
        end
        %% Assign local variables to global ones
        sigpresentreal(siriter, ebnoiter, iter) = sigpresreal;
        sigpresentest(siriter, ebnoiter, iter) = sigpres;
        if (sigpres == 1 && sigpresreal == 1)
            delayerr(siriter, ebnoiter, iter) = (sig2delay ~= delayest);
            if delayerr(siriter, ebnoiter, iter) == 0
                bersgen(siriter, ebnoiter, iter) = synchgenber;
                berstand(siriter, ebnoiter, iter) = synchstandber;
                pgsguess(siriter, ebnoiter, iter) = pgbase;
                rotfacguess(siriter, ebnoiter, iter) = rotfac;
                pgsreal(siriter, ebnoiter, iter) = pathG2(sig2delay+1);
                numiters(siriter, ebnoiter, iter) = iters;
                out3ber(siriter, ebnoiter, iter) = out3ber1;
                jmdber11(siriter, ebnoiter, iter) = out1ber1;
                jmdber12(siriter, ebnoiter, iter) = out2ber1;
                jmdber21(siriter, ebnoiter, iter) = out1ber2;
                jmdber22(siriter, ebnoiter, iter) = out2ber2;
                jmdber31(siriter, ebnoiter, iter) = out1ber3;
                jmdber32(siriter, ebnoiter, iter) = out2ber3;
                jmdber41(siriter, ebnoiter, iter) = out1ber4;
                jmdber42(siriter, ebnoiter, iter) = out2ber4;
            end
        end
        prog = prog + 1/(length(sirs)*length(ebnos)*numiter);
        waitbar(prog, wB, strcat(int2str(floor(prog*100)), '%'));
      end
    end
end
```

```
toc;

%% Multi-SIR
temp = sum(~isnan(bersgen), 3);

% 5-SIR Channel Gain MSE
figure;
% This manipulation gets around the data being NaN'd if the interferer was
% never actually sent or simply not detected.
temp2 = abs((pgsguess - pgsreal).^2);
temp2(isnan(temp2)) = 0;
pgerr = sum(temp2, 3) ./ temp;

hold all, grid on;
plot(ebnos, pgerr(1, :), '-s', ebnos, pgerr(2, :), '-^', ...
    ebnos, pgerr(3, :), '-d', ebnos, pgerr(4, :), '-o', ...
    ebnos, pgerr(5, :), '-v', ...
    'LineWidth', 2', 'MarkerSize', 10)
xlabel('E_b/N_o'); ylabel('Channel Gain Mean-Square Error')
title('Joint Synchronization-Detection Channel Gain Estimation Error');
legend('SIR = -6dB', 'SIR = -3dB', 'SIR = 0dB', 'SIR = 3dB', 'SIR = 6dB');
set(gcf, 'Color', 'w');
hold off;

% 5-SIR Frequency Estimation Error
figure;
rotfacerr = abs(rotfacguess - rotfacreal)*180/pi;
freqerr = (rotfacerr / ts) / 360 / 1e3;
freqerr(isnan(freqerr)) = 0;
freqerr2 = sum(freqerr, 3) ./ temp;

hold all, grid on;
title('Joint Synchronization-Detection Frequency Estimation Error');
plot(ebnos, freqerr2(1, :), '-s', ebnos, freqerr2(2, :), '-^', ...
    ebnos, freqerr2(3, :), '-d', ebnos, freqerr2(4, :), '-o',...
    ebnos, freqerr2(5, :), '-v', ...
    'LineWidth', 2', 'MarkerSize', 10)
xlabel('E_b/N_o'); ylabel('Frequency Error (KHz)')
legend('SIR = -6dB', 'SIR = -3dB', 'SIR = 0dB', 'SIR = 3dB', 'SIR = 6dB');
hold off;
% 5-SIR Genetic Algorithm Plot
figure;
hold all;
bersgen2 = bersgen;
bersgen2(isnan(bersgen2)) = 0;
bersgen3 = sum(bersgen2, 3) ./ temp;
grid on;

plot( ...
    ebnos, bersgen3(1, :), '-s', ebnos, bersgen3(2, :), '-^', ...
    ebnos, bersgen3(3, :), '-d', ebnos, bersgen3(4, :), '-o', ...
    ebnos, bersgen3(5, :), '-v', ...
    'LineWidth', 2', 'MarkerSize', 10);
```

```
title('Joint Synchronization-Detection BER');
legend('SIR = -6dB', 'SIR = -3dB', 'SIR = 0dB', 'SIR = 3dB', 'SIR = 6dB');
xlabel('E_b/N_o'); ylabel('BER')
set(gca ,'yscale','log');
hold off;
%% Single-SIR JMD Plot
% This plot works if only a single SIR is simulated. In this case, the
% plots comparing the theoretical and practical performance of the JMD
% algorithm will be plotted. This is currently commented out because more
% than a single SIR is selected to be simulated.


% figure;
% jmdber111 = jmdber11;
% jmdber111(isnan(jmdber111)) = 0;
% jmdber112 = sum(jmdber111, 3) ./ temp;
%
% jmdber121 = jmdber12;
% jmdber121(isnan(jmdber121)) = 0;
% jmdber122 = sum(jmdber121, 3) ./ temp;
%
% jmdber211 = jmdber21;
% jmdber211(isnan(jmdber211)) = 0;
% jmdber212 = sum(jmdber211, 3) ./ temp;
%
% jmdber221 = jmdber22;
% jmdber221(isnan(jmdber221)) = 0;
% jmdber222 = sum(jmdber221, 3) ./ temp;
%
% jmdber311 = jmdber31;
% jmdber311(isnan(jmdber311)) = 0;
% jmdber312 = sum(jmdber311, 3) ./ temp;
%
% jmdber321 = jmdber32;
% jmdber321(isnan(jmdber321)) = 0;
% jmdber322 = sum(jmdber321, 3) ./ temp;
%
% jmdber411 = jmdber41;
% jmdber411(isnan(jmdber411)) = 0;
% jmdber412 = sum(jmdber411, 3) ./ temp;
%
% jmdber421 = jmdber42;
% jmdber421(isnan(jmdber421)) = 0;
% jmdber422 = sum(jmdber421, 3) ./ temp;
%
% hold on;
% x = plot( ...
%     ebnos, jmdber112, '--s', ebnos, jmdber122, ':s', ...
%     ebnos, jmdber212, '--d', ebnos, jmdber222, ':d', ...
%     ebnos, jmdber412, '--v', ebnos, jmdber422, ':v', ...
%     'LineWidth', 2', 'MarkerSize', 10, 'Color', [0 0 0]);
% xlabel('E_b/N_o'); ylabel('BER')
% title('BERs for Joint Detection Algorithm, SIR = 1dB');
% set(gca ,'yscale','log');
```

```
%
% leg = legend('Prime, No Known', 'Interferer, No Known',...
%     'Prime, Freq Known', 'Interferer, Freq Known', ...
%      'Prime, Both Known', 'Interferer, Both Known');
% set(leg, 'Location', 'southwest');
% set(gcf, 'Color', 'w');
% grid on;
% hold off;


close(wB);
```

8.2.2 Delay Estimation Plotter

Simplified version of the system plotter that focuses on the detection algorithm. Returns the delay estimation rates of all three detectors as error rate plots. Note that as the required threshold for the blind detector was not explored, performance for that detector is poor.

```
%% Andrew Apollonsky
close all force;
clc;
clear all;

%% Simulation Parameters
len1 = 12; % Training length
len2 = 60; % Frame length (discounting training)
m1 = 4;
m2 = 4;

ebnos = [0:4:20];
sirs = [-6 -3 0 3 6];
numiter = 20000;
thresh = 0;

ts = 4e-6; %x/s
freq = 2.4; %(GHz)
maxDopp = 1;
rotfacreal = .05;
realfreqoffset = rotfacreal /(2*pi) / ts / 1e3; %KHz

%% Object Initialization
bpskmod = comm.BPSKModulator(0);
bpskdemod = comm.BPSKDemodulator(0);
qam4mod = comm.RectangularQAMModulator(4);
qam4demod = comm.RectangularQAMDemodulator(4);
qam16mod = comm.RectangularQAMModulator(16);
qam16demod = comm.RectangularQAMDemodulator(16);
qam64mod = comm.RectangularQAMModulator(64);
qam64demod = comm.RectangularQAMDemodulator(64);
qam256mod = comm.RectangularQAMModulator(256);
```

```
qam256demod = comm.RectangularQAMDemodulator(256);

if m1 == 2
    mod1 = bpskmod;
    demod1 = bpskdemod;
elseif m1 == 4
    mod1 = qam4mod;
    demod1 = qam4demod;
elseif m1 == 16
    mod1 = qam16mod;
    demod1 = qam16demod;
end

if m2 == 2
    mod2 = bpskmod;
    demod2 = bpskdemod;
elseif m2 == 4
    mod2 = qam4mod;
    demod2 = qam4demod;
elseif m2 == 16
    mod2 = qam16mod;
    demod2 = qam16demod;
end

g1 = 1;

delayerr1 = nan(length(sirs), length(ebnos), numiter);
delayerr2 = nan(length(sirs), length(ebnos), numiter);
delayerr3 = nan(length(sirs), length(ebnos), numiter);
sigpresentest1 = nan(length(sirs), length(ebnos), numiter);
sigpresentest2 = nan(length(sirs), length(ebnos), numiter);
sigpresentest3 = nan(length(sirs), length(ebnos), numiter);
sigpresentreal = nan(length(sirs), length(ebnos), numiter);

wB = waitbar(0,'0%');
prog = 0;
%% Calculations
[evm1, evm2, var1, var2] = errcalc(sirs, ebnos + 10*log10(log2(m1)),...
    1e5, mod1, demod1, mod2);
[evm11, evm22, var11, var22] = errcalc(-10:10, ebnos + 10*log10(log2(m1))...
    , 1e5, mod1, demod1, mod2);
tic;
for siriter = 1:length(sirs) % Iterate over SIRs
    sir2 = sirs(siriter) - 10*log10(log2(m1)) + 10*log10(log2(m2));
    g2 = 10^(-sir2/20);
    for ebnoiter = 1:length(ebnos) % Iterate over SNRs
            SNR = ebnos(ebnoiter) + 10*log10(log2(m1));
            for iter = 1:numiter % Iterate over Iterations
                sig2delay = randi([1 len2-len1 - 1]); % Generate Delay
                totlen = sig2delay + len1 + len2;

                % Generate signals
                x1 = [randi([0 m1-1], len2, 1); zeros(sig2delay + len1, 1)];
                x2 = [zeros(sig2delay, 1); randi([0 1], len1, 1);...
```

```
    randi([0 m2-1], len2, 1)];

% Modulate
y1 = [step(mod1, x1(1:len2)); zeros(sig2delay + len1, 1)];
y2 = [zeros(sig2delay, 1); ...
    step(bpskmod, x2(sig2delay+1:sig2delay+len1)); ...
    step(mod2, x2(sig2delay+len1+1:end))];

% Generate path gains
pathG1 = [ones(len2, 1) * exp(rand(1) * 2*pi*1i) * g1; ...
    zeros(len1 + sig2delay, 1)];
pathG2 = [zeros(sig2delay, 1); ones(len1 + len2, 1) *...
    exp(rand(1) * 2*pi*1i) * g2];

% Introduce frequency offset
for t = sig2delay+1:totlen
    pathG2(t) = pathG2(t) * exp((t-sig2delay-1) ...
        * 1i * rotfacreal);
end

% Apply path gains to data
rx1 = pathG1 .* y1;
rx2 = pathG2 .* y2;

% Add AWGN
rx1 = awgn(rx1, SNR, mean(abs(y1(1:len2)).^2));

% Sum signals, or no interferer
if rand(1) > .5
    rxSig = rx1 + rx2;
    sigpresreal = 1;
else
    rxSig = rx1;
    sigpresreal = 0;
end

% Find if signal present, and if so, find delay
[delayest1, sigpres1] = ...
    delayfind(rxSig, mod1,... % Both SIR and SNR
    demod1, pathG1, len2, evm1(siriter, ebnoiter), ...
    evm2(siriter, ebnoiter), 0);

[delayest2, sigpres2] = ...
    delayfind2(rxSig, mod1,... % Neither SNR nor SIR
    demod1, pathG1, len2, 0);

[delayest3, sigpres3] = delayfind(rxSig, mod1,... % SNR Only
    demod1, pathG1, len2, mean(evm11(:, ebnoiter)),...
    mean(evm22(:, ebnoiter)), 0);
%% Assign to Global
sigpresentreal(siriter, ebnoiter, iter) = sigpresreal;
sigpresentest1(siriter, ebnoiter, iter) = sigpres1;
sigpresentest2(siriter, ebnoiter, iter) = sigpres2;
sigpresentest3(siriter, ebnoiter, iter) = sigpres3;
```

```
                % Detect errors
                if (sigpres1 == 1 && sigpresreal == 1)
                    delayerr1(siriter, ebnoiter, iter) = ...
                        (sig2delay ~= delayest1);
                end
                if (sigpres2 == 1 && sigpresreal == 1)
                    delayerr2(siriter, ebnoiter, iter) = ...
                        (sig2delay ~= delayest2);
                end
                if (sigpres3 == 1 && sigpresreal == 1)
                    delayerr3(siriter, ebnoiter, iter) = ...
                        (sig2delay ~= delayest3);
                end


            end
            prog = prog + 1/(length(sirs)*length(ebnos));
            waitbar(prog, wB, strcat(int2str(round(prog*100)), '%'));
    end
end
toc;
%% Plot
%% Delay Error for Different Detectors
% SNR and SIR
temp2 = sum(~isnan(delayerr1), 3);
delayerr12 = delayerr1;
delayerr12(isnan(delayerr1)) = 0;
delayerr13 = sum(delayerr12, 3) ./ temp2;
temp5 = delayerr13;

figure;
hold all;
plot(ebnos, temp5(1, :), '-s', ebnos, temp5(2, :), '-d',...
    ebnos, temp5(3, :), '-x', ...
    ebnos, temp5(4, :), '-^', ebnos, temp5(5, :), '-+',...
    'LineWidth', 2', 'MarkerSize', 10);
title(...
    'Delay Error Rate vs E_b/N_o and SIR  requiring SNR and SIR Knowledge');
xlabel('E_b/N_o'); ylabel('Delay Error Rate')
grid on;
legend('SIR = -6dB', 'SIR = -3dB', 'SIR = 0dB', 'SIR = 3dB', 'SIR = 6dB')
set(gca ,'yscale','log');
hold off;


% Neither SNR nor SIR
temp2 = sum(~isnan(delayerr2), 3);
delayerr22 = delayerr2;
delayerr22(isnan(delayerr2)) = 0;
delayerr23 = sum(delayerr22, 3) ./ temp2;
temp5 = delayerr23;

figure;
```

```
hold all;
plot(ebnos, temp5(1, :), '-s', ebnos, temp5(2, :), '-d',ebnos, ...
    temp5(3, :), '-x', ...
    ebnos, temp5(4, :), '-^', ebnos, temp5(5, :), '-+',...
    'LineWidth', 2', 'MarkerSize', 10);
title(...
    'Delay Error Rate vs E_bN_o and SIR requiring no SNR or SIR Knowledge');
xlabel('E_b/N_o'); ylabel('Delay Error Rate')
grid on;
legend('SIR = -6dB', 'SIR = -3dB', 'SIR = 0dB', 'SIR = 3dB', 'SIR = 6dB')
set(gca ,'yscale','log');
hold off;

% SNR Only
temp2 = sum(~isnan(delayerr3), 3);
delayerr32 = delayerr3;
delayerr32(isnan(delayerr3)) = 0;
delayerr33 = sum(delayerr32, 3) ./ temp2;
temp5 = delayerr33;

figure;
hold all;
plot(ebnos, temp5(1, :), '-s', ebnos, temp5(2, :), '-d', ...
    ebnos, temp5(3, :), '-x', ...
    ebnos, temp5(4, :), '-^', ebnos, temp5(5, :), '-+',...
    'LineWidth', 2', 'MarkerSize', 10);
title('Delay Error Rate vs E_bN_o and SIR requiring only SNR Knowledge');
xlabel('E_b/N_o'); ylabel('Delay Error Rate')
grid on;
legend('SIR = -6dB', 'SIR = -3dB', 'SIR = 0dB', 'SIR = 3dB', 'SIR = 6dB')
set(gcf, 'Color', 'w');
set(gca ,'yscale','log');
hold off;

close(wB);
```

8.2.3 ROC Plotter

Modified plotter that only iterates through either SNR or SIR. This makes it convenient for generating ROC characteristics for the three receivers.

```
%% Andrew Apollonsky
close all force;
clc;
clear all;

%% Simulation Parameters
len1 = 12; % Training length
len2 = 60; % Frame length (discounting training)
m1 = 4;
m2 = 4;
```

```
% WARNING: This script only supports one of ebnos or sirs being a vector.
% Other must be a scalar. Make sure legends, etc are appropriately labeled.
ebnos = [0:4:20]; % Modify the 'SNRS'
sirs = [5]; % SIRS

threshes = [0 logspace(-6, 0, 600)]; % Range of thresholds
numiter = 1000;
ts = 4e-6; % Sampling period
freq = 2.4; %(GHz)
maxDopp = 1;
rotfacreal = .05; % Rotation factor
realfreqoffset = rotfacreal /(2*pi) / ts / 1e3; %KHz

%% Object Initialization
bpskmod = comm.BPSKModulator(0);
bpskdemod = comm.BPSKDemodulator(0);
qam4mod = comm.RectangularQAMModulator(4);
qam4demod = comm.RectangularQAMDemodulator(4);
qam16mod = comm.RectangularQAMModulator(16);
qam16demod = comm.RectangularQAMDemodulator(16);

if m1 == 2
    mod1 = bpskmod;
    demod1 = bpskdemod;
elseif m1 == 4
    mod1 = qam4mod;
    demod1 = qam4demod;
elseif m1 == 16
    mod1 = qam16mod;
    demod1 = qam16demod;
end

if m2 == 2
    mod2 = bpskmod;
    demod2 = bpskdemod;
elseif m2 == 4
    mod2 = qam4mod;
    demod2 = qam4demod;
elseif m2 == 16
    mod2 = qam16mod;
    demod2 = qam16demod;
end

g1 = 1;
sigpresentest = nan(max(length(ebnos), length(sirs)),...
    numiter, length(threshes));
sigpresentest2 = nan(max(length(ebnos), length(sirs)),...
    numiter, length(threshes));
sigpresentest3 = nan(max(length(ebnos), length(sirs)),...
    numiter, length(threshes));
sigpresentreal = nan(max(length(ebnos), length(sirs)), numiter);
sigpres = nan(length(threshes), 1);
sigpres2 = nan(length(threshes), 1);
```

```
sigpres3 = nan(length(threshes), 1);

wB = waitbar(0,'0%');
prog = 0;
%% Calculations
% Calculate expected error magnitudes
[evm1, evm2, var1, var2] = ...
    errcalc(sirs, ebnos + 10*log10(log2(m1)), 1e5, mod1, demod1, mod2);
[evm11, evm22, var11, var22] = ...
    errcalc(-10:10, ebnos + 10*log10(log2(m1)), 1e5, mod1, demod1, mod2);
tic;

% Select which of ebnos and sirs is vector
if length(ebnos) > 1
    vars = ebnos;
else
    vars = sirs;
end

% Loop over ebnos/sirs
for variter = 1:length(vars)
    %Properly allocate all variables depending on which is vector
    if length(ebnos) > 1
        SNR = ebnos(variter) + 10*log10(log2(m1));
        sir2 = sirs - 10*log10(log2(m1)) + 10*log10(log2(m2));
        ebnoiter = variter;
        siriter = 1;
    else
        SNR = ebnos + 10*log10(log2(m1));
        sir2 = sirs(variter) - 10*log10(log2(m1)) + 10*log10(log2(m2));
        siriter = variter;
        ebnoiter = 1;
    end
    g2 = 10^(-sir2/20);

    for iter = 1:numiter % Iterate over iterations
        sig2delay = randi([1 len2-len1 - 1]); % Generate delay
        totlen = sig2delay + len1 + len2;

        % Generate data
        x1 = [randi([0 m1-1], len2, 1); zeros(sig2delay + len1, 1)];
        x2 = [zeros(sig2delay, 1); randi([0 1], len1, 1);...
            randi([0 m2-1], len2, 1)];

        % Modulate data
        y1 = [step(mod1, x1(1:len2)); zeros(sig2delay + len1, 1)];
        y2 = [zeros(sig2delay, 1); ...
            step(bpskmod, x2(sig2delay+1:sig2delay+len1)); ...
            step(mod2, x2(sig2delay+len1+1:end))];

        % Generate channel gains
        pathG1 = [ones(len2, 1) * exp(rand(1) * 2*pi*1i) *...
            g1; zeros(len1 + sig2delay, 1)];
        pathG2 = [zeros(sig2delay, 1); ones(len1 + len2, 1) * ...
```

```
            exp(rand(1) * 2*pi*1i) * g2];

        % Introduce frequency-offset-caused rotation
        for t = sig2delay+1:totlen
            pathG2(t) = pathG2(t) * exp((t-sig2delay-1) * 1i * rotfacreal);
        end

        % Apply path/channel gains to data
        rx1 = pathG1 .* y1;
        rx2 = pathG2 .* y2;

        % Add AWGN
        rx1 = awgn(rx1, SNR, mean(abs(y1(1:len2)).^2));

        % Sum signals
        if rand(1) > .5
            rxSig = rx1 + rx2;
            sigpresreal = 1;
        else
            rxSig = rx1;
            sigpresreal = 0;
        end

        % Find if signal present, and if so, delay. For every threshold.
        for threshiter = 1:length(threshes)
            sigpres(threshiter) = delayfind(rxSig, mod1,...
            demod1, pathG1, len2, evm1(siriter, ebnoiter), ...
            evm2(siriter, ebnoiter), threshes(threshiter));
            sigpresentest(variter, iter, threshiter) = sigpres(threshiter);

            [~, sigpres2(threshiter)] = delayfind2(rxSig, mod1,...
            demod1, pathG1, len2, threshes(threshiter));
            sigpresentest2(variter, iter, threshiter) = ...
                sigpres2(threshiter);

            sigpres3(threshiter) = delayfind(rxSig, mod1,...
            demod1, pathG1, len2, mean(evm11(:, ebnoiter)), ...
            mean(evm22(:, ebnoiter), 1), threshes(threshiter));
            sigpresentest3(variter, iter, threshiter) = sigpres3(threshiter);

        end
        %% Assign to Global
        sigpresentreal(variter, iter) = sigpresreal;
        prog = prog + 1/(length(sirs)*length(ebnos) * numiter);
        waitbar(prog, wB, strcat(int2str(round(prog*100)), '%'));
    end

end
toc;
%% Plot
repreal = repmat(sigpresentreal, [1 1 length(threshes)]);
fa1 = sum((repreal == 0) .* ...
    (sigpresentest == 1), 2)./sum(repreal == 0, 2); % False Alarm
ta1 = sum((repreal == 1) .* ...
```

```
    (sigpresentest == 1), 2)./sum(repreal == 1, 2); % True Alarm
fa2 = squeeze(fa1);
ta2 = squeeze(ta1);

fa12 = sum((repreal == 0) .* ...
    (sigpresentest2 == 1), 2)./sum(repreal == 0, 2); % False Alarm
ta12 = sum((repreal == 1) .* ...
    (sigpresentest2 == 1), 2)./sum(repreal == 1, 2); % True Alarm
fa22 = squeeze(fa12);
ta22 = squeeze(ta12);

fa13 = sum((repreal == 0) .* ...
    (sigpresentest3 == 1), 2)./sum(repreal == 0, 2); % False Alarm
ta13 = sum((repreal == 1) .* ...
    (sigpresentest3 == 1), 2)./sum(repreal == 1, 2); % True Alarm
fa23 = squeeze(fa13);
ta23 = squeeze(ta13);

% Plot Detector I ROC (SIR and SNR)
figure;
plot(fa2.', ta2.', 'LineWidth', 2');
legend('E_b/N_o = 0dB', 'E_b/N_o = 4dB', 'E_b/N_o = 8dB',...
    'E_b/N_o = 12dB', 'E_b/N_o = 16dB', 'E_b/N_o = 20dB');
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('SIR/SNR Knowledge Detection Scheme');

% Plot Detector II ROC (Not SNR or SIR)
figure;
plot(fa22.', ta22.', 'LineWidth', 2');
legend('E_b/N_o = 0dB', 'E_b/N_o = 4dB', 'E_b/N_o = 8dB',...
    'E_b/N_o = 12dB', 'E_b/N_o = 16dB', 'E_b/N_o = 20dB');
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('No Knowledge Detection Scheme');

% Plot Detector III ROC (SNR only)
figure;
plot(fa23.', ta23.', 'LineWidth', 2');
legend('E_b/N_o = 0dB', 'E_b/N_o = 4dB', 'E_b/N_o = 8dB', ...
    'E_b/N_o = 12dB', 'E_b/N_o = 16dB', 'E_b/N_o = 20dB');
xlabel('False Positive Rate');
ylabel('True Positive Rate');
title('SNR Knowledge Detection Scheme');

close(wB);
```

8.2.4 Expected Error Magnitude Generator

Generates the expected error vector magnitude given the constellations of the two signals. Returns a matrix in SNR and SIR if given vector SNR and SIR arguments.

```
function [evm1, evm2 ,var1, var2] = ...
        errcalc(sirs, snrs, iters, mod1, demod1, mod2)
    m1 = length(constellation(mod1));
    m2 = length(constellation(mod2));
    g1 = 1;

    % Generate data
    x1 = randi([0 m1-1], iters, 1);
    x2 = randi([0 m2-1], iters, 1);

    % Modulate Data
    y1 = step(mod1, x1);
    y2 = step(mod2, x2);

    evm1 = zeros(length(sirs), length(snrs));
    evm2 = zeros(length(sirs), length(snrs));
    for k = 1:length(sirs) % Loop over required SIRs
        g2 = 10^(-sirs(k)/20);
        parfor m = 1:length(snrs) % Loop over required SNRs
            % Generate phase offsets for y1. y2 done later.
            pathG1 = exp(rand(iters, 1) * 2*pi*1i) * g1;
            z1 = y1 .* pathG1; % Multiply
            z2 = y2 .* exp(rand(iters, 1) * 2*pi*1i) * g2; % phase for y2
            rx1 = awgn(z1, snrs(m), 'measured'); % Apply AWGN for no interf.
            rx2 = awgn(z1, snrs(m), 'measured') + z2; % AWGN for interf.
            % Expected error vector magnitude with no interferer
            evm1(k, m) = mean(abs(rx1./pathG1 - ...
                step(mod1, step(demod1, rx1./pathG1))));
            % Expected error vector magnitude with interferer
            evm2(k, m) = mean(abs(rx2./pathG1 - ...
                step(mod1, step(demod1, rx2./pathG1))));
            % Variance of error vector magnitude with no interferer
            var1(k, m) = var(abs(rx1./pathG1 - ...
                step(mod1, step(demod1, rx1./pathG1))));
            % Variance of error vector magnitude with interferer
            var2(k, m) = var(abs(rx2./pathG1 - ...
                step(mod1, step(demod1, rx2./pathG1))));
        end
    end

end
```

8.2.5 Interference Detector I

This script implements both the interference detection algorithm which uses both SNR and SIR and the one that only uses SNR. The only difference between the two implementations is that when the exact SIR is unknown, a mean of the expected error over several SIRs is passed to the algorithm instead of the value for a particular SIR.

```
function [delay, sig] = delayfind(rx, mod1, demod1, pathG1, len2, ...
        evm1, evm2, thresh)
```

```
    k = rx(1:len2)./pathG1(1:len2); % Divide received signal by channel gain
    k2 = step(mod1, step(demod1, k)); % De and re-modulate
    evmvec = abs(k2 - k); % Find error magnitude
    msqerr = zeros(len2+1, 1);
    for delayiter = 1:len2 % Find error using each delay
        msqerr(delayiter) = mean(...
            [abs((evmvec(1:delayiter) - evm1).^2); ...
            abs((evmvec(delayiter + 1:min(len2, delayiter+12)) - evm2).^2)]);
    end
    msqerr(end) = mean(abs((evmvec(1:len2)-evm1).^2));%Error with no interf.
    if msqerr(end) - min(msqerr(1:len2)) < thresh; % Make judgement
        sig = 0;
        delay = nan(1);
    else
        [~, delay] = min(msqerr(1:len2));
        sig = 1;
    end

end
```

8.2.6 Interference Detector II

This script implements the interference detection algorithm which uses neither the SNR nor the SIR. The ideal threshold which to use to detect whether the interference is there or not under varying conditions was not derived, so using this algorithm in its full capacity would be error-prone.

```
function [delay, sig] = delayfind2(rx, mod1, demod1, pathG1, len2, thresh)
    k = rx(1:len2)./pathG1(1:len2); % Divide received signal by channel gain
    k2 = step(mod1, step(demod1, k)); % De and re-modulate
    evmvec = abs(k2 - k); % Find error magnitude
    dist = zeros(len2, 1);
    for delayiter = 1:len2 % Find distance between mean error mags
        dist(delayiter) = abs(...
            mean(evmvec(1:delayiter)) - ...
            mean(evmvec(delayiter + 1:min(len2, delayiter+12)))...
            );
    end
    if max(dist) < thresh; % If it's never large enough, no interferer
        sig = 0;
        delay = nan(1);
    else
        [~, delay] = max(dist); % Otherwise, interferer where it's largest
        sig = 1;
    end
end
```

8.2.7 Genetic Synchronization Algorithm

This script implements the genetic algorithm that synchronizes the interferer while decoding the primary signal. Due to this being by far the single most computationally-intensive part of the system, great pains were taken to vectorize the script and make it efficient, at the cost of readability.

```
function [out1, PG2, iters, pgbase, rotfac] = gen_al(data, d2, mod1, pg)
const = constellation(mod1);
m = length(const);
len = length(data);

% Convergence Checking
checking = 0; % Set to 1 for convergence checking
numcheck = 5; % Number of solutions out to check
iternum = 25;

% Other parameters
numsols = 60; % This is m
maxpg = 10^(10/20); % Maximum power of interferer relative to primary
minpg = 10^(-10/20);
maxfreq = pi/10; % Maximum rotation of constellation per symbol period

% Initialization
iter = true;
iters = 1;
sols = zeros(numsols, len + 2);
rotvec = zeros(numsols^2 + numsols, len);
for k = 1:numsols % Initialize Path Gain, Phase-Change-Per-Turn, Symbols x
    sols(k, :) = [newpg(minpg, maxpg, 1, 1) ...
        newang(maxfreq, 1, 1) const(randi(m, [1 len]))).'];
end

while iter == true
    % Cross-mix the vector of solution vectors with itself -- generate M
    % from K
    ranum = rand(len+2, numsols, numsols); % Generates random matrix
    combos = zeros(len+2, numsols, numsols); % Will be the new M matrix
    sols2 = permute(sols, [3 1 2]); % Permute for easier indexing

    for k = 1:2 % Continuous case--corresponds to phase change&channel gain
        %30%Pick from first set
        ranum2 = ranum(k, :, :) < .3; %
        combos(k, :, :) = combos(k, :, :) + ranum2 .* ...
            repmat(sols2(:, :, k), [1 1 numsols]);

        % 30% Pick from second
        ranum2 = .3 < ranum(k, :, :) & ranum(k, :, :) < .6;
        combos(k, :, :) = combos(k, :, :) + ranum2 .* ...
            repmat(permute(sols2(:, :, k), [1 3 2]), [1 numsols 1]);

        % 20% Take mean
```

```
        ranum2 = .6 < ranum(k, :, :) & ranum(k, :, :) < .8;
        combos(k, :, :) = combos(k, :, :) ...
            + ranum2 .*(repmat(permute(sols2(:, :, k), ...
            [1 3 2]), [1 numsols 1]) + repmat(sols2(:, :, k), ...
            [1 1 numsols])) / 2;

        % 20% Generate new value
        ranum2 = .8 < ranum(k, :, :);
        if k == 1 % If channel gain, generate new channel gain
            combos(k, :, :) = combos(k, :, :) + ranum2 .* ...
                permute(newpg(minpg, maxpg, numsols, numsols), [3 1 2]);
        else % If phase rotation, generate new phase rotation
            combos(k, :, :) = combos(k, :, :) + ranum2 .* ...
                permute(newang(maxfreq, numsols, numsols), [3 1 2]);
        end
    end
    for k = 3:len+2 % For discrete variables -- the x's
        ranum2 = ranum(k, :, :) < .45; % 45% chance copying from 1st
        combos(k, :, :) = combos(k, :, :) + ranum2 .* ...
            repmat(sols2(:, :, k), [1 1 numsols]);

        ranum2 = .45 < ranum(k, :, :) & ranum(k, :, :) < .9; % 45% from 2nd
        combos(k, :, :) = combos(k, :, :) + ranum2 .* ...
            repmat(permute(sols2(:, :, k), [1 3 2]), [1 numsols 1]);

        ranum2 = .9 < ranum(k, :, :); % 10% chance of new value
        combos(k, :, :) = combos(k, :, :) + ranum2 .* ...
            permute(const(randi(m, numsols)), [3 1 2]);
    end
%% ctd
    % Serialize matrix M into single massive vector of solution vectors.
    % Include old K matrix.
    comboser = [reshape(combos, 2 + len, [], 1) sols.'];

    % For each solution in the massive vector, generate a rotation vector
    for t = 1:len
        rotvec(:, t) = exp(1i*comboser(2, :)).^(t-1);
    end

    % Evaluating fitness of every solution in massive vector of solution
    % vectors
    fits = mean((abs(repmat(data, 1, numsols^2 + numsols) - ...
        comboser(3:end, :) .* repmat(pg, 1, numsols^2 + numsols) - ...
        repmat(comboser(1, :), len, 1) .* rotvec.' .* ...
        repmat(d2, 1, numsols^2 + numsols))), 1).';

    % Sort by fitness, pick the m best solutions
    [~, bestind] = sort(fits, 1, 'ascend');
    % New K vector of solution vectors
    sols = comboser(:, bestind(1:numsols)).';

    % Checking for convergence
    best = sols(1:numcheck, :);
    decmod = repmat(mode(best(:, 3:end), 1), numcheck, 1);
```

```
    decindx = mean(mean(decmod == best(:, 3:end)));
    if checking == 1
        if decindx >= symbthresh
            iter = false;
        else
            iters = iters + 1;
        end
    else
        iters = iters + 1;
    end

    if iters == iternum
        break;
    end
end
out1 = sols(1, 3:end).'; % Output the x
rotfac = sols(1, 2); % Output the best rotation factor estimate
pgbase = sols(1, 1);% Output the best channel gain estimate
% Combination of rotation factor & gain
PG2 = rotvec(bestind(1), :).* sols(1, 1);
end

% Generate new channel gain estimate
function [pgout] = newpg(ming, maxg, y, x)
    pgout = (ming + (maxg-ming).*rand(y, x)) * exp(1i * 2*pi*rand(y, x));
end

% Generate new rotation factor estimate
function [angout] = newang(maxrot, y, x)
    angout = -maxrot + (2*maxrot * rand(y, x));
end
```

8.2.8 Joint Detector

Implementation of the minimum-distance joint detection algorithm for underdetermined MIMO systems.

```
function [out1, out2] = JMD(rx, mod1, mod2, pathG1, pg2)
    % Initialization
    m1 = length(constellation(mod1));
    m2 = length(constellation(mod2));
    out1 = zeros(length(rx), 1);
    out2 = zeros(length(rx), 1);
    mini = zeros(length(out1), m1, m2);

    for q = 0:(m1-1) % Look through potential values of y1
        for p = 0:(m2-1) % Potential values of y2
            % Compute the expected sum of the two for each combo
            mini(:, q+1, p+1) = pathG1.*step(mod1, q) + pg2.*step(mod2, p);
        end
    end
```

```
    % Find the absolute distance for every combo
    minimat = abs(repmat(rx, [1 m1 m2]) - mini);

    for p = 1:length(rx)
        % Find smallest index for each point in time
        [~, minind] = min(reshape(minimat(p, :, :), [], 1));
        % Get associated indeces for y1 and y2
        [~, a, b] = ind2sub(size(minimat(p, :, :)), minind);
        out1(p) = a-1; % assign y1
        out2(p) = b-1; % assign y2
    end
end
```

Bibliography

[1] J. Lee, D. Toumpakaris, and W. Yu, "Interference Mitigation via Joint Detection," in *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 6, June 2011

[2] K. Xu, M. Gerla, S. Bae, "How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks," in *Global Telecommunications Conference*, vol. 1, November 2002

[3] `http://standards.ieee.org/news/2014/ieee_802_11ac_ballot.html`

[4] `http://www.ieee802.org/11/Reports/tgad_update.htm`

[5] S. Keene, J. Carruthers. "Improved Error Correction in Wireless LANs Using Erasures Decoding with Collision Localization," in *Global Telecommunications Conference*, pp. 4451-4455, Nov. 2007

[6] G. Judd and P. Steenkiste. "Using Emulation to Understand and Improve Wireless Networks and Applications," in *NSDI*, 2005.

[7] S. Gollakota, D. Katabi, "ZigZag Decoding: Combating Hidden Terminals in Wireless Networks," *Proc. ACM Sigcomm*, pp. 159170, Aug. 2008.

[8] S. Rahman, Y. Li, B. Vucetic. "An Iterative ZigZag Decoding for Combating Collisions in Wireless Networks," in *IEEE Communications Letters*, vol. 14, no. 3, March 2010

[9] S. Jafar, S. Shamai. "Degrees of Freedom Region of the MIMO X Channel," in *IEEE Transactions on Information Theory*, vol. 54, no. 1, January 2008.

[10] O. Ayach, S. Peters, R. Heath. "The Practical Challenges of Interference Alignment," in *IEEE Wireless Communications*, pp. 35-42, February 2013.

[11] S. Jafar, V. Cadambe. "Interference Alignment and Degrees of Freedom of the K-User Interference Channel" in *IEEE Transactions on Information Theory*, vol. 54, no. 8, August 2008.

[12] J. Lee, W. Kim, et al. "An Experimental Study on the Capture Effect in 802.11a Networks", 2007.

[13] J. Spall, *Introduction to Stochastic Search and Optimization. Estimation, Simulation, and Control*. John Wiley & Sons, Inc, 2003.

[14] Andrea Goldsmith, *Wireless Communications*. New York, NY: Cambridge University Press, 2005.

[15] John G. Proakis and Masoud Salehi, *Digital Communications*, 5th Ed. New York, NY: McGraw-Hill, 2008.

[16] Michael Rice, *Digital Communications: A Discrete-Time Approach*, Upper Saddle River, NJ: Pearson Prentice Hall, 2009.

[17] Alain Sibille, Claude Oestges, Alberto Zanella, *MIMO: From Theory to Implementation*, Burlington, MA: Elsevier, 2011.

[18] David Tse, Pramod Viswanath, *Fundamentals of Wireless Communication*, Cambridge University Press, 2005.

[19] Yong Soo Cho, Jaekwon Kim, Won Young Yang, Chung-Gu Kang, *MIMO-OFDM Wireless Communications with MATLAB*, Singapore: IEEE Press, 2010.