MONTE CARLO BASED CHANNEL ESTIMATION AND EVALUATION FOR 5G MILLIMETER-WAVE SYSTEMS

by

Alec Weiss

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Antennas and Wireless Communications).

Golden, Colorado

Date _____

Signed: _____

Alec Weiss

Signed: _____

Dr. Atef Elsherbeni
Thesis Advisor

Golden, Colorado

Date _____

Signed: _____

Dr. Peter Aaen
Professor and Head
Department of Electrical Engineering Engineering

ABSTRACT

Current wireless communications systems utilizing frequencies below 6 GHz are quickly becoming unable to meet consumer needs due to limited bandwidth. The need for ever increasing data throughput for wireless systems is pushing systems to begin using higher millimeter-wave frequencies to provide frequency bandwidths that are orders of magnitude larger than those operating at the lower frequencies. Operating at millimeter-wave frequencies brings along new challenges that do not exist at lower frequencies such as mass produced hardware that is physically smaller to overcome the larger free space loss which can bring increase uncertainty in channels and hardware components. The uncertainty increases in these systems must be corrected by channel estimators to correctly receive data. Channel estimators are effected by the increased hardware error and their efficacy is therefore reduced. This in turn increases the error in received data and reduces the throughput of a communications system.

A multitude of channel estimation techniques have been developed in an attempt to reduce bit errors in the received data and maximize the system throughput. Methods can be as basic as performing division and averaging of known transmitted values to estimate the channel. Other channel estimators have been designed using techniques ranging from machine learning to optimization and minimization. None of these estimators use responses and uncertainties of hardware components that are typically provided by manufacturers. By directly using known hardware responses and any corresponding uncertainties, a channel estimator can improve its estimation.

The research outlined here first provides a framework for the consistent evaluation and comparison of estimators in millimeter-wave systems and second develops an improved channel estimator that leverages known hardware responses and their uncertainties. The comparison between estimators is important to select the best estimator for a scenario before deployment of hardware to save time and money. Without a framework for this comparison, there is no way to directly test channel estimators without implementing and deploying them into a physical system. The developed framework provides common communications metrics in phased array architectures and propagation environments to evaluate how different channel estimators perform. This framework is then used to evaluate a novel Monte Carlo augmented channel estimator. This channel estimator uses repeat simulations of the known hardware responses and their uncertainties to provide an improved channel estimation. This channel estimator is directly compared to other commonly implemented channel estimators to demonstrate how it can be used in communications systems to reduce overall bit errors in received data and increase system throughput.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

3^{rd} Generation Partnership Project 3GPP

Angle of Arrival . . . . . AoA

Base Station . . . . . BS

Basic Linear Algebra Subroutines . . . . . BLAS

Binary Phase Shift Keying . . . . . BPSK

Central Processing Unit . . . . . CPU

Channel Estimation . . . . . CE

Channel Frequency Response . . . . . CFR

Channel Impulse Response . . . . . CIR

Computational Electromagnetics . . . . . CEM

Conventional BeamForming . . . . . CBF

Cyclic-Prefix Orthogonal Frequency Division Multiplexing . . . . . CP-OFDM

Discrete Fourier Transform . . . . . DFT

Extremely High Frequency . . . . . EHF

Fast Fourier Transform . . . . . FFT

Graphics Processing Unit . . . . . GPU

Independent and Identically Distributed . . . . . IID

Institute of Electrical and Electronics Engineers . . . . . IEEE

Inter-Symbol-Interference . . . . . ISI

Just In Time Compilation . . . . . JIT

Least Squares . . . . . LS

Linear Algebra Package . . . . . LAPACK

Linear Minimum Mean Square Error . . . . . LMMSE

Math Kernel Library . . . . . MKL

Millimeter-Wave . . . . . mmWave

Minimum Mean Square Error . . . . . MMSE

Monte Carlo Augmented Channel Estimator . . . . . MCCE

New Radio . . . . . NR

Orthogonal Frequency Division Multiplexing . . . . . OFDM

Power Delay Profile . . . . . PDP

Quadrature Amplitude Modulation . . . . . QAM

Quadrature Phase Shift Keying . . . . . QPSK

Random Access Memory . . . . . RAM

Synthetic Aperture Measurements with UnceRtainty and Angle of Incidence . . . . . SAMURAI

Third Generation Partnership Project 3GPP

User Equiptment . . . . . UE

Wireless Local Area Network . . . . . WLAN

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Atef Elsherbeni, for helping me in all of my academic persuits and continually driving me to complete and defend this dissertation. I would also like to thank Jeanne Quimby and all of my colleagues at NIST for the training and insight they have provided me over my multiple years there. Without them, this work would not be possible.

This paper is dedicated to my friends, family, and most of all my future wife, Rebecca. You have stuck beside me and never stopped encouraging me to reach my goals, even while we quarantined together.

CHAPTER 1
INTRODUCTION TO MILLIMETER-WAVE COMMUNICATION SYSTEMS

Millimeter-wave communications systems are becoming more commonplace in the world of commercial communications systems and the standards that define them. Many previous communications systems rely on relatively low frequencies typically below about 6 GHz. These systems are able to reach vast distances due to the low losses at lower frequencies allowing better wireless coverage from a single point of access. In today's connected world, the bandwidth available at lower frequencies cannot support the desired throughput. By utilizing much higher frequencies near 30 GHz and above, vast amounts of wireless spectrum are able to be utilized to significantly increase the throughput of wireless communications systems. These frequencies have wavelengths measured in millimeters and are therefore commonly referred to as Millimeter-Waves (mmWaves).

Next generation commercial wireless communications systems leverage millimeter-waves and electrically large phased arrays to support use cases such as enhanced mobile broadband. Channel estimators provide the corrections necessary for propagation channel and phased array hardware to receive the transmitted signal correctly and achieve a high quality of service. Phased arrays in mmWave communication systems utilize electrically larger phased arrays than at microwave frequencies bringing more significant hardware uncertainties. These uncertainties can include variations in expected responses due to manufacturing, hardware noise, and tighter tolerances. To better estimate and correct the transmitted signal, a new Monte Carlo-augmented channel estimator was developed for millimeter-wave orthogonal frequency division multiplexing systems by incorporating the predefined known responses of system hardware and their uncertainties. Using metrics such as mean square error, error vector magnitude, and bit error ratio, the estimator is compared against current channel estimators for different phased-array configurations and propagation channels. A framework has also been developed to provide consistent testing and comparison the developed Monte Carlo estimator and other previously developed channel estimators.

1.1 Basics of Millimeter-Wave Communication Systems

Millimeter-Wave technology has been utilized in radar and point to point communications links for decades [1]. Until recently, the technology and lack of funding available for mobile millimeter-wave communications systems have made them unfeasible. Now, many commercial entities and standardization bodies are working to bring along the next generation of mobile communications systems.

Two of the most promising modern specifications are the IEEE 802.11ad (and its new counterpart 802.11ay) standard for WLAN [2], and the 3^{rd} Generation Partnership Projects (3GPP) 5G new radio

(NR) specification [3]. Each of these standards relies in part on the usage of millimeter-waves to provide wireless connections with throughputs topping multiple gigabits per second. While the work in this dissertation is commonly generalized and will therefore apply to both standards, the 3GPP 5G NR specification will be the focus of this dissertation.

1.2 Basics of OFDM Communications Systems

Orthogonal frequency division multiplexing (OFDM) has long been a method of efficient data transmission and has been a topic of study for many decades [4, 5]. OFDM utilizes multiple orthogonal frequency domain subcarriers. This means that in the frequency domain, data is parallelized across multiple carrier frequencies or subcarriers. The waveforms at each of these subcarriers is then shaped such that it has a value of zero at each of the adjacent subcarriers. This ensures that there is no interference between subcarriers. An example of this with four subcarriers shaped with a sinc function can be seen in Figure 1.1. In practice, the orthogonal channels are typically achieved by generating a frequency domain signal and calculating the inverse Fourier transform of the signal, allowing generation of signals in software. Each of these discrete signals is then shaped in the time domain by applying a filter such as a root-raised cosine (RRC) to preserve orthogonality. This has been the topic of many studies and is well covered for 5G systems [6].

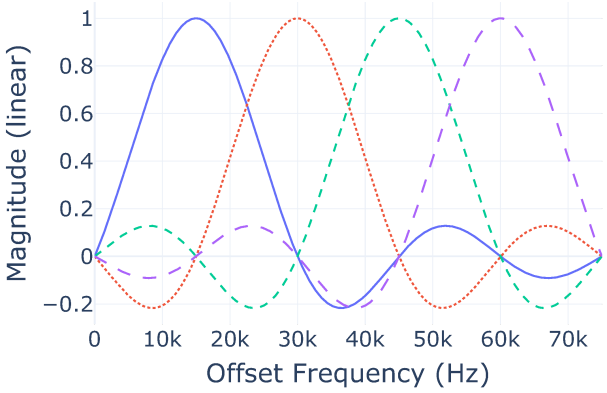


Figure 1.1 Example of four orthogonal subcarriers in the frequency domain shaped with a sinc function at intervals of 15 kHz. Note that at the center frequency (peak) of each subcarrier, all other subcarriers have a value of 0.

The use of orthogonal subcarriers allows modulated data to be sent in parallel making for very efficient data transmission. The transmission of data can be made even more efficient through the usage of different modulation schemes such as quadrature amplitude modulation 256 (256-QAM). The 5G modulation schemes specified by the Third Generation Partnership Project (3GPP) are listed in [7, p. 13-14].

While many implementations of OFDM exist, it is likely that future 5G systems will employ a cyclic prefix OFDM (CP-OFDM) [6] which is also widely in use today. This method of OFDM provides resistance inter-symbol-interference (ISI) while also allowing for more error in symbol alignment when receiving the OFDM symbol. The use of CP-OFDM may change between implementation which will effect the time domain signal and its resistance to ISI. Changes in this implementation do not change the core operation of an OFDM system.

1.2.1 Typical OFDM Implementations

A full block diagram of a typical OFDM system for 5G mm-wave communications can be seen in Figure 1.2. This is similar to the block diagrams of OFDM systems found in [8, p. 361] and [9]. In this scheme,

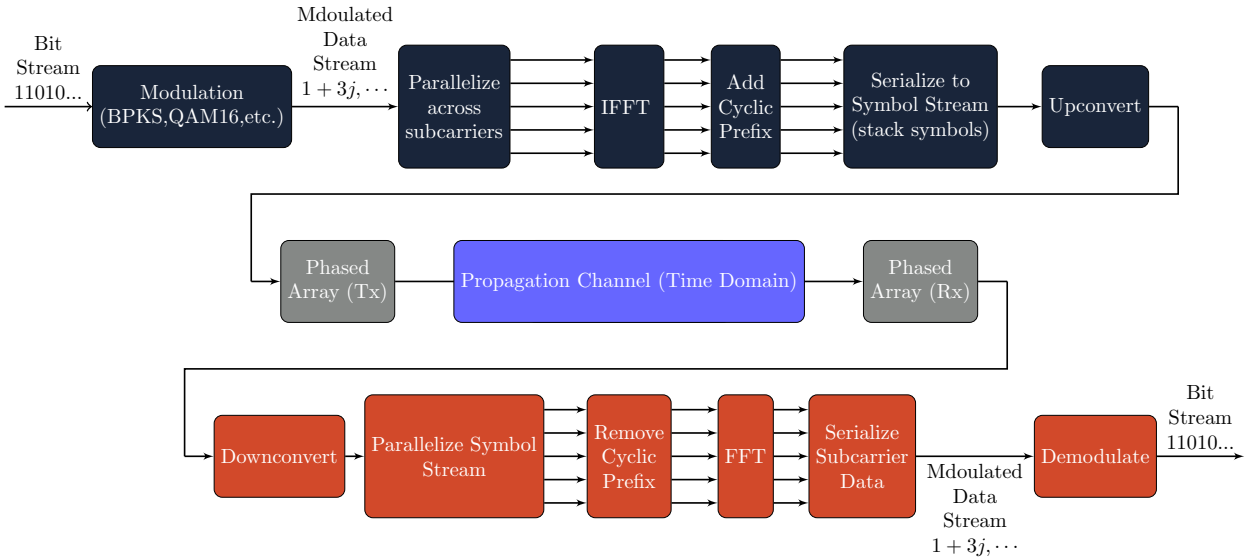


Figure 1.2 Block diagram for a 5G millimeter-wave OFDM communications system. This includes both time and frequency domain operations

an input data stream ($D_{bits} = [1, 1, 0, 1, 0, ...]$) is first modulated. This can be multiple modulations ranging from binary phase shift keying (BPSK) all the way to 256 quadrature amplitude modulate (256-QAM). The modulation scheme utilized is dependent on the error rate of the received data and the 5G NR specification allows for the usage of these modulation schemes and many in between (such as 16-QAM). Again, for a full list of the 5G NR modulation schemes and their definitions, see [7, p. 13-14]. The modulation of the data maps the bitstream to a constellation of points on a complex plane. Once the data has been modulated, it is represented by a data stream of complex numbers ($D_{cplx} = [1 + 3j, 2 - 1j, ...]$). These complex numbers are then parallelized across OFDM subcarriers. For the 5G specification the subcarriers are spaced at frequency spacings given as $\delta f = 15\ kHz * 2^{\mu}$ where μ is between 0 and 4 [10, p. 9]. This set of frequency domain sub-

carriers is then transformed to a time domain signal via a inverse fast Fourier transform (IFFT). Depending on the OFDM scheme, the diagram may vary after this point. For 5G OFDM systems, a cyclic prefix (CP) is added to each symbol (each symbol is comprised of a complex value at each subcarrier). This cyclic prefix allows for a receiver to have a better chance of recovering a symbol without inter-symbol-interference (ISI) by prepending part of the end of the time domain symbol. These time domain signals are then serialized to make a single data stream. From here, the signal is converted to an analog waveform and upconverted to a carrier frequency. For 5G mm-wave systems in the US, these frequencies are typically either in the 24 GHz [11], 28 GHz [12], or 39/47 GHz bands [13]. Each of these bands provides up to 400 MHz of bandwidth.

With an upconverted OFDM signal typical 5G OFDM mm-wave systems typically then utilize phased arrays to overcome the higher propagation loss at mmWave frequencies. For full communications systems this brings new challenges. The phased arrays can be steered toward a user to provide a higher gain to overcome the increased path loss. Larger arrays are feasible at mmWaves when compared to lower frequency communications due to the smaller wavelength. The smaller wavelength makes electrically large phased arrays with hundreds or thousands of elements possible. Many mmWave communications implementations may also use phased arrays on both the transmit and receive ends, which introduce dynamic beam-steering requirements and even more hardware noise due to the large number of components and smaller required tolerances.

On the receive side of an OFDM system, the data stream is down-converted to a baseband frequency and converted into a digital signal. Each symbol is then extracted from the data stream. The cyclic prefix is then removed and a fast Fourier transform (FFT) is performed to bring the symbol back into a frequency domain representation. From this frequency domain representation, the data on each subcarrier for each symbol can be extracted and serialized into a modulated data stream which again consists of a stream of complex numbers representing points on a complex plane ($D_{cplx} = [1+3j, 2-1j, ...]$). In real communications systems, the channel effects (i.e. all effects between transmitting and receiving data that adjust the values of the transmitted data) can be estimated and the data must then be corrected to account for loss and channel fading to reduce the error rate in the data transmission. More on this can be found in 1.4. After this correction the data can then be demodulated with the same scheme used for modulation, and a data stream is output from the receiver ($D_{bits} = [1, 1, 0, 1, 0, ...]$).

1.3 Challenges of Millimeter-Wave Communications Systems

New challenges arise when moving from lower frequencies up to mmWaves. One of the largest factors effecting the utility of mmWaves is the increased free space loss. From Frii's transmission equation it is shown that the received power is affected as $\frac{1}{f^2}$ where f is the operating frequency causing power received

to be drastically lower than that of a lower frequency communication system. These losses only increase as obstacles and weather are added into the mix [14, 15, 16]. To overcome the added propagation loss, large phased arrays are becoming commonplace to achieve higher gain and allow the ability to steer the beam to a receiving device.

The short wavelength at mmWaves makes electrically large phased arrays with many elements physically and commercially viable. The components in these antennas must be mass produced with tighter manufacturing tolerances and reduced power requirements when compared to lower frequency systems [17]. Imperfect phased array hardware calibrations can introduce systematic shifts in hardware expected hardware responses and lead to bit errors during communication [18]. The high bandwidths proposed for 5G mmWave communications also inherently increase thermal noise [19, 20] increasing the total noise of the communication system. The combination of higher noise, increased production output, and decreased tolerances all increase the uncertainties in each of the hardware components which in turn can directly affect the integrity of received data in a communications system.

1.4 Channel Estimation for Millimeter-Wave Communications Systems

Alongside the high bandwidths and complex hardware used in mmWave communication systems, modern modulation and multiplexing schemes must also be utilized to achieve the desired throughputs. These multiplexing schemes must be adjusted to balance between throughput and error rate. System performance can therefore be improved by correcting data for anything that changes a transmitted value before it reaches the receiver. Anything that introduces these changes is considered part of the channel. Channel estimation can be used to account for hardware and propagation channel responses and uncertainties from a transmitter to a receiver and therefore allow for higher density modulations without increasing bit errors. Consequently, increasing the modulation density with decreased bit errors increases the overall throughput of a communications system.

This correction is performed after estimating the response of the channel through channel estimation or channel estimator (CE). A typical channel estimation problem for OFDM systems can be written as

$$y_f = x_f h_f + n_f \tag{1.1}$$

where y is the received signal, x is the transmitted data and h is the complex channel response [21]. The value at each frequency x_f is typically known by both the transmitter and receiver in an OFDM system by sending a set of known pilot tones on known specific OFDM subcarriers. The receiver receives these pilot tones which can be represented by a frequency domain value, y_f, across each of the OFDM subcarriers. The value y_f is the known pilot tone, x_f, multiplied by the channel frequency response with added noise from the system

represented by n_f. For OFDM systems, many times it is easier to deal with the signals in frequency domain due to the nature of the frequency multiplexing. These pilot tones can be interleaved with the transmission of data in time (as a function of each OFDM symbol being sent), frequency (across subcarriers), or both [22]. Figure 1.3 shows an example of how pilot tones help estimate a channel. Each of the pilot tone subcarriers shown in green provide known values. These known values are used to calculate the response $h(f)$ at these subcarriers. Interpolation can then be used to estimate $h(f)$ at all of the non-pilot subcarriers (in red) to correct received data. Through these interleaved pilot tones, many different algorithms have been developed to estimate the channel response h. Two of the most common algorithms for channel estimation are least squares (LS) estimator and linear minimum mean squared error (LMMSE) estimator.

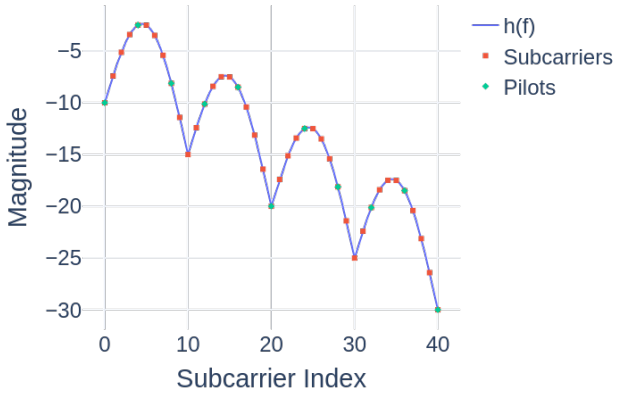


Figure 1.3 Example pilot tone distribution across subcarriers in an OFDM communications system.

1.4.1 Least Squares and Minimum Mean Squared Error Estimators

The least squares and minimum mean squared error have been used for OFDM channel estimation for decades. Both of these methods are well described for OFDM systems in [21] with variants that utilize mathematical techniques such as singular value decomposition (SVD) and some that vary the configuration of pilot tones to improve channel estimation [23, 24, 22]. Similar to the response mentioned before, each of these estimation techniques can be used to solve for the channel h in

$$\mathbf{y} = X\mathbf{h} + \mathbf{n} \tag{1.2}$$

where y is the received value, h is the channel response at each frequency, n is the noise in the system, and X is a matrix whose diagonal components are the symbols transmitted on each of the pilot tones [21].

Least squares estimation is the simplest of the two techniques mentioned. Using equation (1.2), the least squares estimate of h is given by minimizing $\|\mathbf{y} - X\mathbf{h}\|^2$ which is given by

$$X^{-1}\mathbf{y} = \mathbf{h} \tag{1.3}$$

where X^{-1} is the inverse of the matrix X [21]. Because X in this case is a diagonal matrix of the complex value of each transmitted pilot tone, the inverse is simply the diagonal matrix containing values of

$$X^{-1} = \begin{cases} \frac{1}{x_i} & i \\ 0 & otherwise \end{cases} \tag{1.4}$$

which then provides values of h as the element-wise division of y and $diag(X)$. This gives the much simpler expression

$$h = \left[\frac{y_1}{x_{11}}, \frac{y_2}{x_{22}}, \cdots, \frac{y_N}{x_{NN}} \right]^T \tag{1.5}$$

where each pilot tone subcarrier index $i = 1, 2, \cdots, n$ corresponds to a different subcarrier frequency. For many cases, this implementation is ideal as it provides a very low computational complexity. Depending on timing and power constraints, LS estimator may be the only applicable method for some systems. If, for example, A channel is changing fast enough to effect each OFDM symbol differently (e.g. in a fast moving car), LS may be the only feasible method to compute a channel estimate in a short enough time period.

Minimum mean squared error estimation is more complex than LS, but takes into account pre-determined auto-correlations and noise of the channel. Basics of minimum mean squared error are well studied and detailed in many references such as [25]. This method allows predefined correlations between variables to be utilized to minimize the error of the estimate of a variable. For the case of channel estimation, this means using a predefined cross frequency correlation of the sub-carriers to provide a better estimate of the channel. This is an extension onto the LS estimator by

$$h_{LMMSE} = Ah_{ls} \tag{1.6}$$

where the matrix A is a weighting matrix derived from the pre-defined auto-correlation of the frequency response of a channel. A is defined as

$$A = R_{hh}(R_{hh} + \sigma_n^2(XX^H)^{-1})^{-1} \tag{1.7}$$

as given by [26] where R_{hh} is the auto-correlation of the entire channel (propagation channel and hardware responses) that must be known before calculation. It should be recognized that for most implementations this method may be restrictive for a couple of reasons. First, for time and power sensitive applications the LMMSE takes many more operations to calculate than the LS does. As previously mentioned, the MMSE

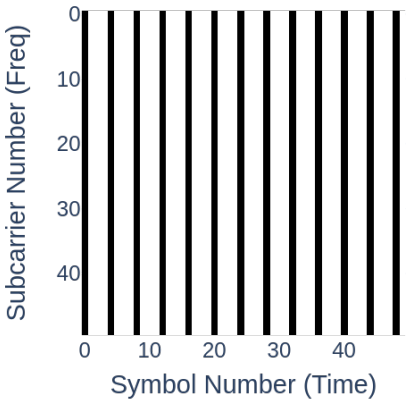
calculation also requires channel information to be known before estimation is performed. Specifically, the auto-covariance R_{hh} of the channel must be known along with the variance of the noise σ_n^2. For many applications, this knowledge may not be possible to know beforehand. While simplifications to reduce these calculations have been developed using methods such as SVD [23], the computational requirements will always be greater than LS, and the necessity of a-priori knowledge still exists. For these reasons, many systems in application will still use an LS scheme for channel estimation.

1.4.2 Estimation of Sub-carriers without Pilot Tones

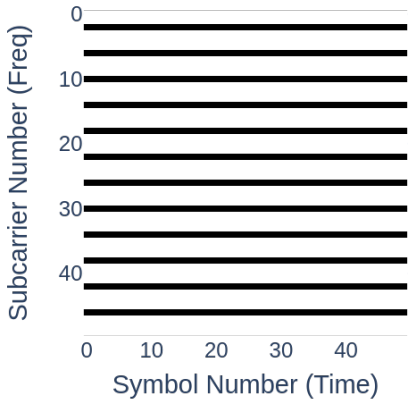
The channel estimators described in the previous section try to accurately estimate the complex channel response h. As mentioned previously, typically the pilot tones will be interleaved with data and therefore the channel estimators can only estimate the response at given frequencies f_p where the pilot tones exist. Using only this data, the channel responses at the rest of the subcarrier frequencies must then be estimated. Many techniques can be utilized for estimating these intermediate frequencies. The most basic of these methods is to utilize a block type pilot transfer [24] as seen in Figure 1.4(a). For channels that are static or whose responses vary slowly, full OFDM symbols can be sent with only pilot tones. The response calculated at each of the subcarriers is then used as a correction for the next OFDM symbols that consist only of data. This setup requires no extra steps past initial channel estimation, but does assume that the channel response is static from one OFDM symbol to the next. For systems that change more rapidly with respect to the length of an OFDM symbol, a simple and commonly used method is to interpolate to find the channel responses at the frequencies between f_p after sending a comb type pilot tone arrangement [24, 22] as seen in Figure 1.4(b). Using interpolation assumes that the number of pilot tones used is adequate to describe the features present in the frequency domain fading of the channel. If too few pilot tones are used, the estimation of channel responses subcarrier frequencies may be incorrect, and the error rate of the data transmission will quickly increase. If too many pilot tones are used, the data rate of the system will be decreased as fewer subcarriers will be available for data transfer. While other methods for subcarrier response recovery have been developed, they are not the focus on this work. For both comb and block type transfers, the error of the channel response estimation is dependent on the estimation of the channel on subcarriers containing pilot tones as any error in these estimations will propagate through to estimation of frequencies without pilot tones.

1.4.3 Improved Channel Estimation Techniques

5G mmWave channel estimators must account for new and unique large-scale array architectures, more complicated hardware responses, and wideband channels. A number of unique large-scale array architectures



(a) Block type pilot arrangement



(b) Comb type pilot arrangement

Figure 1.4 Visualizations of pilot tone arrangements. Shaded sections denote where pilot tones are across frequency

may use hardware with increased uncertainties at each array element due to manufacturing, power, and cost constraints [27]. These uncertainties include factors such as increased thermal noise and uncertainties from manufacturing with lower tolerances. These uncertainties are then incoherently combined producing non-Gaussian output noise distributions and larger systematic offsets. Extremely wideband channels at mmWave frequencies also introduce features that may affect channel estimation, which may not be noticeable at lower bandwidths and frequencies [16]. These mmWave propagation channels are also subject to rapid changes due to environmental effects like rain and humidity [28].

Many methods have been developed to solve some of these new challenges. Work has been done on channel estimation [29, 30] exploiting spatial components of the large-scale phased array architectures. These estimators leverage the capability of large phased arrays to produce multiple beams with very small beamwidths. These separate beams are spatially diverse and can therefore be used to generate information to improve channel estimators that would otherwise be unknown. Other works have focused on other estimation methods using machine learning techniques [31]. The works in [32, 33] utilize neural networks to try and provide fast estimation that provides improved performance over the LS estimator in the presence of noise. Furthermore, denoising strategies to decrease the noise for a variety of channel estimation techniques is discussed in [34].

Other researchers have utilized compressed sensing and basis pursuit methods for channel estimation such as in [35, 36, 37]. These and other closely related methods typically will utilize dictionaries of basis sets that are matched to the channel estimation to provide a good estimation of the channel. These dictionaries are typically predefined or trained after the setup of the system in an environment. This type of method

focuses on improving throughput by using compressed sensing to estimate responses on the subcarriers in between the pilot tones. Many times simple methods such as linear interpolation are used for the estimation between subcarriers but with these compressed sensing methods, fewer pilot tones are required to accurately reconstruct the response across the frequency domain. This in turn requires a lower density of pilot tones and allows for higher throughput.

To reduce the effect of hardware uncertainties on channel estimation, there is a variety of work on the correction of phase, magnitude, IQ imbalance, and other uncertainties and noise in OFDM systems [38, 39, 40]. While each of these reduces the overall system error to provide a better channel estimation, nothing ever provides a perfect correction.

1.5 Contributions

The main contributions of the work described in this dissertation are the development of an evaluation framework to improve the consistency of the evaluation of channel estimators in mmWave systems along with the development and testing of a novel channel estimator that leverages Monte Carlo methods for improved channel estimation.

1.5.1 Improved Evaluation of Millimeter-Wave Channel Estimators

A framework was developed to provide a consistent evaluation of mmWave channel estimators for large scale phased array systems. This channel estimator evaluator builds upon previous simulations tools for 5G channel estimators to provide a simple and consistent tool for comparing the efficacy of channel estimators in a variety of propagation channels and phased array configurations with varying hardware components. This estimator evaluator adds capabilities where previous works fall behind in the comparison and evaluation of channel estimators in 5G mmWave communications systems. The design, implementation and testing of this framework can be found in chapter 2.

1.5.2 Improving Channel Estimators with Monte Carlo Methods

A new channel estimator that utilizes Monte Carlo methods for improving channel estimation is also presented. Typical current 5G channel estimators have a variety of strengths, but most do not account for the large variability seen in 5G mmWave systems due to increased noise, production variability, and decreased power requirements. The Monte Carlo augmented channel estimator (MCCE) presented here accounts for known hardware responses and hardware error distributions to provide a reduced error in the channel estimate. This in turn increases the maximum throughput of a system by producing fewer bit errors in the output signal. More details on the implementation and results when using this new channel estimator can be found in chapter 3.

CHAPTER 2

SIMULATION AND EVALUATION OF CHANNEL ESTIMATORS IN 5G MILLIMETER-WAVE SYSTEMS

Providing a consistent and rigorous comparison between channel estimators before deployment can save both time and money. This comparison requires simulating in a variety of propagation channels with different hardware components in large-scale phased arrays. The channel estimator evaluator enables a rigorous and consistent evaluation of different channel estimators in varying mmWave propagation channels and large-scale phased array architectures. No previous tests have provided evaluation to this depth while ensuring consistency across estimators.

2.1 Evaluation of Channel Estimators

A variety of commercial [41], [42] and non-commercial [43, 44, 45] (open source, academic, etc.) 5G simulation tools exist. Many of these tools could be used to evaluate channel estimators by integrating estimators into a simulation of a 5G communication system with varying degrees of success. Each of these tools is successful in achieving the specific purpose for which they were built, but they are incomplete as a 5G channel estimator evaluator for one or more of the following reasons:

- *Hardware Uncertainties* – These tools do not propagate phase array hardware uncertainties into their performance metrics, thus neglecting the effects of non-ideal mmWave hardware on the channel estimator's efficacy.
- *Scalability* – These tools are not designed for easy construction and simulation of large-scale phased arrays. In this case, it is unlikely that the software is optimized for vectorized computation on these phased arrays.
- *Complexity* – These tools are designed as general-purpose software for the simulation of any number of applications. While this provides flexibility in what the software can do, it typically increases the computation times and learning difficulty.
- *Cost* – Open source and tools designed in academia are likely free of charge; more capable commercial software may not be affordable for many wireless engineers and academic researchers.
- *Extensibility* – New features cannot be easily added to the software. This reduces the flexibility of the software for new use cases.

A channel estimator evaluator is needed to fill the gap not covered by the other tools as noted above. Previous authors [46, 47] have developed estimator evaluation software, but are not capable of testing in various wireless communication configurations, including large-scale mmWave phased array architectures and propagation channels similar to those used in 5G communications systems. A fundamentally new aspect of evaluating channel estimators is the quantification of the effect of the increased hardware uncertainties in mmWave systems. The uncertainties associated with hardware components, such as amplifiers and phase shifters, must also be accounted for inside the phased array architectures to capture possible non-Gaussian distributions of the outputs.

By incorporating parts of both propagation channel [42, 43] and circuit focused [41, 45] software, a channel estimator evaluator was developed to allow for fast and in-depth testing of channel estimators for mmWave wireless communications. This evaluator meets the needs not fulfilled by previous evaluation frameworks by leveraging the Python programming language and focusing solely on channel estimator evaluation. It also is scalable for testing estimators with various phased array sizes, hardware components with uncertainties, and propagation channels. A comparison of some of the capabilities of the channel estimator evaluator presented here with other software packages can be found in Table 2.1 and Table 2.2. While most other softwares have additional functionality, none meet the capabilities desired for the evaluation of channel estimators in 5G mmWave systems. The evaluator developed here is unique in its ability to simulate both hardware and channels while maintaining scalability. These and other unique capabilities allow for the testing of configurations like large phased arrays with S-Parameter based components to be steered and simulated to test the effect on a channel estimator, which is typically not possible in other software.

Table 2.1 Comparison of some modeling capabilities of different software packages for evaluating channel estimators.

Name	Hardware Response	Propagation Response	Uncertainties
ADS [41]	Yes	*No*	Yes
HFSS [42]	Yes	Yes	Yes
NYUsim [43]	*No*	Yes	Yes
Vienna LL [45]	Yes	Yes	Yes
This Evaluator	Yes	Yes	Yes

2.2 Design of a Channel Estimator Evaluator

The core of the evaluator is a frequency-domain-based orthogonal frequency division multiplexing (OFDM) simulator. Simulation of OFDM communications systems has been performed for decades [48]. Other implementations have been provided in a variety of programming languages with a variety of capabilities such

Table 2.2 Comparison of some evaluation requirements laid out in section 2.1 for the evaluation of channel estimators.

Name	Scalable	Extensible	Complexity
ADS [41]	*No*	*No*	Medium
HFSS [42]	Yes	*No*	High
NYUsim [43]	N/A	Yes	Low
Vienna LL [45]	Yes	*No*	Medium
This Evaluator	Yes	Yes	Low

as MATLAB [49, 50, 51], Python [52, 53], and with lower level languages such as C/C++ [54]. Simulation has also been performed on a variety of hardware platforms such as graphics processing units (GPUs) [55]. Each of these previously mentioned systems has typically dealt with OFDM communications systems using lower frequencies and bandwidth.

Newer simulations of OFDM communications systems have also addressed simulation for 5G systems such as for a massive MIMO testbed [56] and a 5G extension [44] to the popular NS-3 (network simulator) framework [57]. Work done by previous researchers was extended upon to create an evaluator that meets the goals outlined in the beginning of this section.

For the frequency domain based OFDM simulation, the block diagram from Figure 1.2 is updated to give Figure 2.1 A mixture of analog and digital simulations forms the basis of the channel estimator evaluator to evaluate the performance of an estimator-under-test (EUT) in different propagation channels and wireless communication configurations (e.g., phased array hardware, modulation, etc.). Figure 2.2 provides an overview of operation of the evaluator.

The digital side of the simulation creates, modulates, and packetizes the test data. This data is then passed to an analog simulation to capture the response of hardware and propagation channels. The output of the analog simulation is then passed back to a digital simulation to estimate and correct the channel, depacketize and demodulate data, and generate system-level metrics. The following assumptions are made in the evaluator:

- *No Inter-Symbol-Interference (ISI)* - Each symbol is perfectly received and time aligned preventing ISI. The cyclic prefix is long enough to remove any multipath effects and is perfectly removed and the timing between the transmitter and receiver is perfectly synchronized.

- *Linearity* - All hardware components and all propagation channel effects are assumed to be in a linear regime. Composite frequency domain responses can therefore be obtained through multiplication.

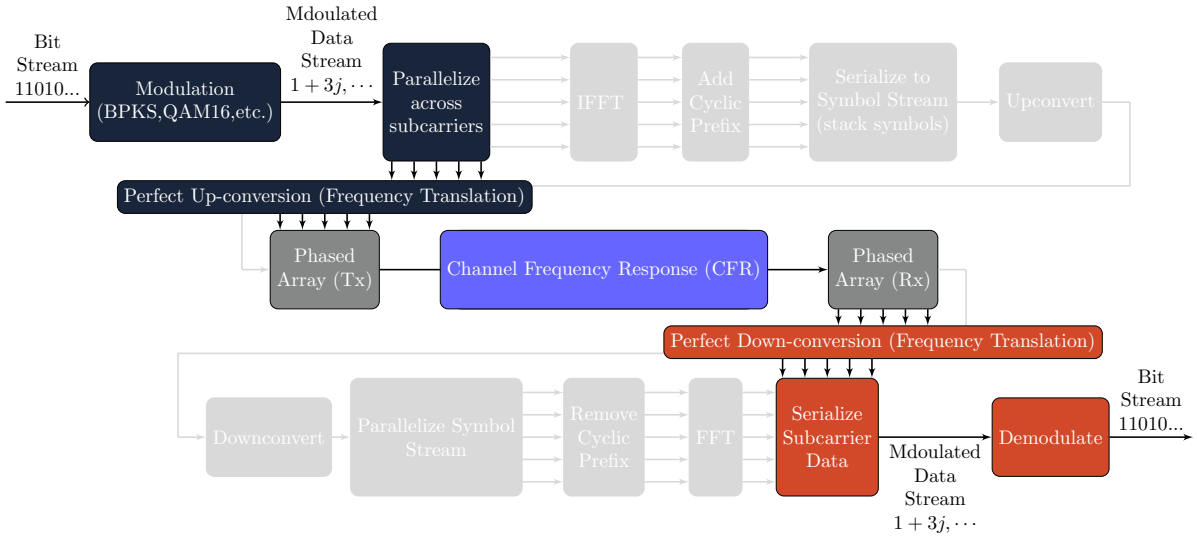


Figure 2.1 Block diagram for a 5G millimeter-wave OFDM communication system frequency domain simulation.

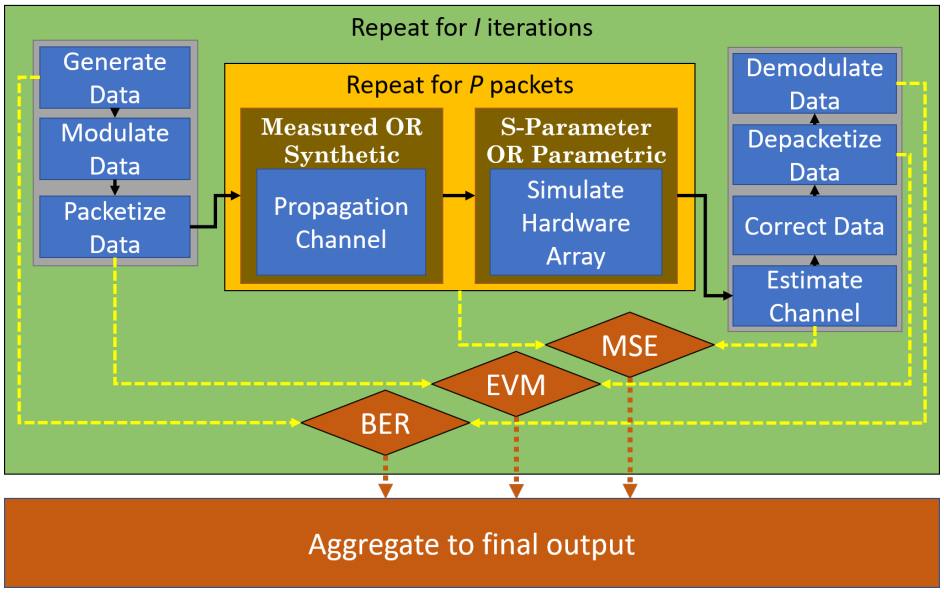


Figure 2.2 Block diagram of the channel estimator evaluator.

- *Perfect Up/Down-Conversion* - Up and down-conversion to/from IF and baseband frequencies is assumed to be ideal. Conversion is performed by translating frequencies as opposed to mixing with a local oscillator (LO) signal. LO noise could still be simulated through non-ideal frequency translation.
- *Perfect Analog $\leftrightarrow$ Digital Conversion* - Analog to digital and digital to analog conversion is assumed to be continuous (no bit discretization) and error free.

These assumptions focus the investigation on hardware component and propagation channel effects when evaluating channel estimators. Future work may systematically remove these assumptions to test their effect on channel estimators as well.

The evaluation accuracy is further improved by capturing uncertainties in hardware components through Monte Carlo methods [58] sampled from predefined distributions. The introduction of uncertainties prevents precomputation of the hardware responses. Therefore, the evaluator must be recalculated every time a new sampled value is used (i.e., every Monte Carlo iteration). While this becomes computationally intensive, these simulations are necessary for an accurate evaluation of the channel estimator.

Performance metrics are used to quantify the efficacy of a channel estimator. These system-level performance metrics include mean squared error (MSE) [58], error vector magnitude (EVM) [59, 60], and bit error ratio (BER) [61]. The MSE is calculated between the channel estimator output and the response from Monte Carlo simulations of the estimated portion of the communication system (e.g. channel and phased array hardware). During demodulation, the EVM is calculated for a given modulation (e.g., 16QAM). Finally, the received data is directly compared to the transmitted data using the BER. The mathematical details of these calculations are provided below in section 2.4.1.

The Python programming language is used for the evaluator. MATLAB and Python can vary in speed depending on the library used and problem being solved, but in many cases the speeds using these languages are very similar as shown in [62, 62]. Python also provides more flexibility with data types and has access to many free libraries. Additionally, it is completely free and was therefore chosen over MATLAB. Using Python also makes the evaluator more accessible to a larger part of the community that may not have access to MATLAB.

2.2.1 Digital Simulation

The evaluator first simulates the digital portion of a wireless OFDM communication system similar to that outlined in section 1.2.1. The evaluator begins by generating input data as a pseudo-random bit sequence. To easily ensure the random data covers all bits, random 8 bit integer values between 0 and 255 are generated. While all bit combinations could be covered with other data types, this is the easiest way to

ensure each bit sequence has an equal chance. An example of this data would look like

$$D_{in} = [64, 12, 240, ..., 117, 85, 2] \tag{2.1}$$

where D_{in} is the input data to transmit through the communications system. This input data is then converted to a bit array, D_{bits}, using the python bitarray library.

This bitstream is then modulated using one of the constellations defined in the 3GPP specification for 5G wireless communication systems [7]. For example, the mapping of bits to a 16-QAM constellation is therefore given by

$$\begin{aligned} d(i) = \frac{1}{\sqrt{10}}\{&(1 - 2D_{bits}(4i))[2 - (1 - 2D_{bits}(4i + 2))] \\ &+ j(1 - 2D_{bits}(4i + 1))[2 - (1 - 2D_{bits}(4i + 3))]\} \end{aligned} \tag{2.2}$$

where each value $d(i)$ is the i^{th} complex symbol mapped to the QAM constellation from the bitstream. These are defined by 3GPP for the following:

- $\frac{\pi}{2}$ BPSK
- BPSK
- QPSK
- 16-QAM
- 64-QAM
- 256-QAM

As the density of the modulation scheme is increased, the modulation equation is much longer as there are more bits per symbol, but the general form of the modulation equation remains the same as that in (2.2). It is also worth noting that if bits per symbol for the modulation technique does not exactly match the number of bits in the bitstream, the end of the bitstream is padded with zeros. Using each of these predefined equations, functions can be written for the mapping of each modulation. Each of these mapping functions is then used to both map the bit stream to a QAM constellation along with creating a dictionary used to unmap the values from the constellation on the receiving end of the simulation. An example of a random bitstream mapped to a 16-QAM constellation using (2.2) is given in Figure 2.3. This is not very interesting as each set of bits is perfectly mapped to its corresponding constellation point although it serves as a good reference of what the data looks like when starting. This modulation step outputs a stream of pseudo-random modulated data, D_{mod}, which may look something like

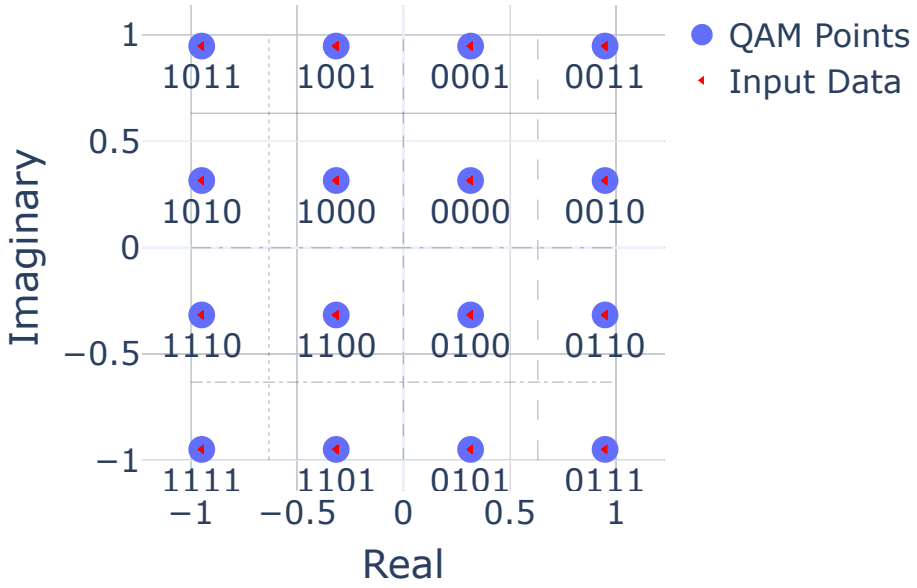


Figure 2.3 A random bitstream mapped to all points on a 16-QAM constellation using (2.2). These values are perfectly mapped and therefore lie in the center of the constellation point.

$$D_{mod} = [0.3 + j0.5, ..., 0.5 + j0.1]. \tag{2.3}$$

The modulated data stream is then packetized across a predefined number of OFDM subcarriers to create a set of OFDM symbols.

While the data is being packed into OFDM symbols, pilot tones are also added to the data. The addition of these pilot tones can be set by the user when calling the function. In order to correctly interleave the pilot tones and data, the pilot tones are first inserted into the subcarriers for each OFDM symbol using a user defined function. This supports flexible pilot tone arrangements over both frequencies and OFDM symbols. This is an important step as it allows for the testing of channel correction algorithms using different pilot tones.

For frequency domain simulation, the modulated and parallelized data only needs be translated across frequencies. This translation is an ideal upconversion that will place all of the baseband energy directly at the carrier frequency (including the removal of any mirror effects). This upconverted signal with pilot tones can be seen in Figure 2.4. One hundred subcarriers are used in this example. Pilot tones each with a value $.95 + .95j$ were placed every 10 subcarriers for this example. In a time domain based OFDM simulation, an IFFT would be performed on the modulated and parallelized data to get a time domain waveform and the waveform would be converted to an analog signal and upconverted for transmission.

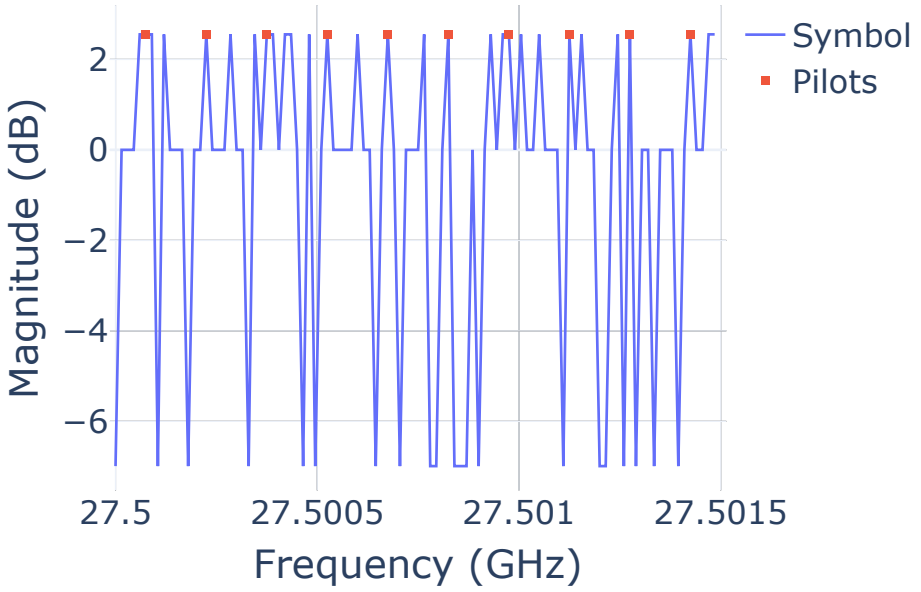


Figure 2.4 Magnitude vs. frequency plot of an OFDM symbol with pilot tones included. This symbol was generated from a random bitstream and modulated using 16-QAM.

The modulated and upconverted OFDM symbols are then passed to the analog simulation described in section 2.2.2. The analog simulation results are then passed back to a digital simulation of a receiver. The output OFDM symbol can be seen in Figure 2.5. The first step in the receiver simulation is to estimate the channel. The received pilot tones are extracted from the received data and passed to an EUT. In order to operate correctly within the evaluator, the EUT must adhere to the implementation given in Listing B.8. The EUT then takes the known input pilot tones and received pilot tones to estimate the effect of the wireless communication hardware and propagation channel on the data. This estimate is used to correct all the received data.

After correction, all the data is extracted from the subcarriers, and rebuilt into a serial data stream. Along with serialization, this step also removes the pilot tones from the subcarriers so they do not end up in the final data stream. Like on the transmit side, this received data can then be plotted onto the QAM constellation for the selected modulation. This can be seen in Figure 2.6. It should be noticed in this plot that the data no longer lies directly on their corresponding constellation points. This is due to imperfections in the estimation of the channel used for correcting the data after reception.

The data is then downconverted (via frequency translation) and demodulated. The demodulation of the data is performed by either a nearest neighbor reverse lookup using a dictionary generated during modulation, or through an arbitrary rounding technique. A reverse lookup can work for any shape of constellation by

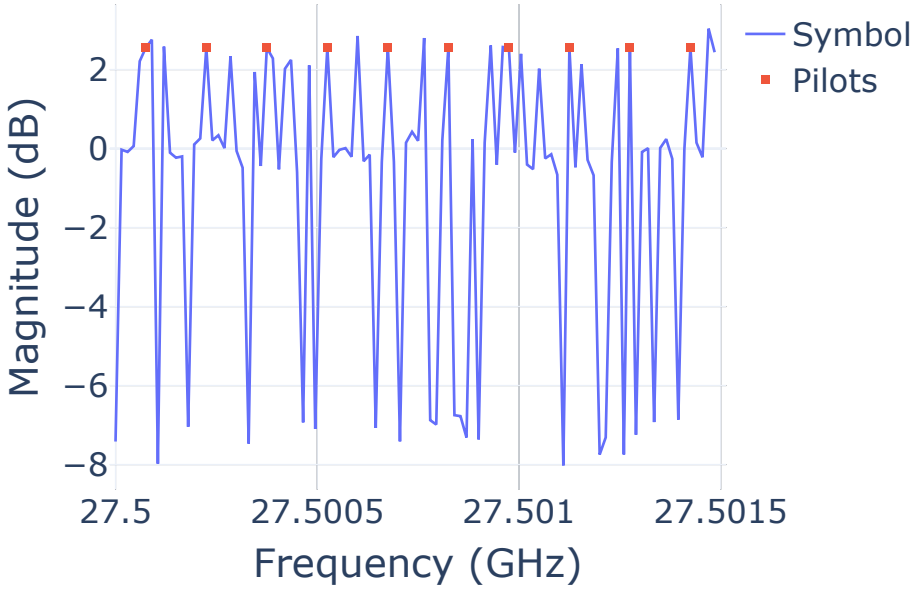


Figure 2.5 Magnitude vs. frequency plot of an OFDM symbol with pilot tones included after simulating transmission through a non line of sight propagation channel and a phased array. This data was corrected with a least squares estimator and linear interpolation from the pilot tones.

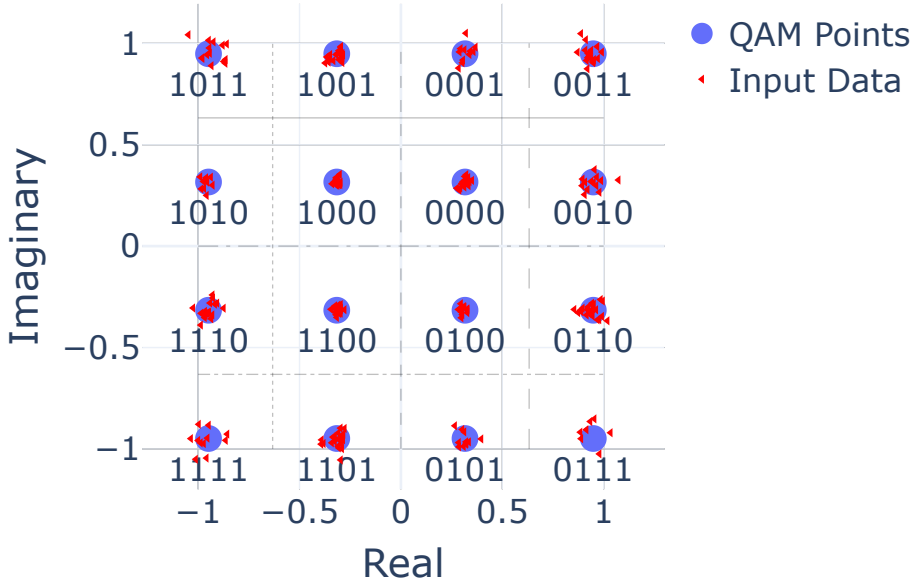


Figure 2.6 Constellation diagram of received data. This data was corrected with a least squares estimator and linear interpolation from the pilot tones.

finding the constellation point with the least distance from the received value. The issue with this method is it requires finding the distance from all constellation points requiring many extra operations. For evenly spaced rectangular constellations (e.g. BPSK, QPSK, 16-QAM, 64-QAM, 256-QAM), A method was developed and utilized to get the same result as the reverse lookup by making each constellation point an integer value and rounding to that integer. This process works as follows, assume a square constellation (e.g. 16-QAM):

1. Translate the received modulated data, $d(i)$ to a constellation with a point at $0 + 0j$ by subtracting an adjustment value α to allow for rounding. This translation can be achieved by letting α equal the first positive value in a constellation. For the 16-QAM constellation given in (2.2), this would make $\alpha = \frac{1}{\sqrt{10}} + j\frac{1}{\sqrt{10}}$. The adjusted modulated data $d_{adj}(i)$ is then given by $d_{adj}(i) = d(i) - \alpha$.

2. Round each adjusted modulated value, $d_{adj}(i)$, to the nearest integer multiple of 2α in each dimension with $\mathfrak{Re}\{d_{rnd}(i)\} = \mathfrak{Re}\{2\alpha\} \lfloor \frac{\mathfrak{Re}\{d_{adj}(i)\}}{\mathfrak{Re}\{2\alpha\}} \rceil$ and $\mathfrak{Im}\{d_{rnd}(i)\} = \mathfrak{Im}\{2\alpha\} \lfloor \frac{\mathfrak{Im}\{d_{adj}(i)\}}{\mathfrak{Im}\{2\alpha\}} \rceil$.

3. Add back the adjustment value α to get the value of the nearest constellation point to the received modulated value $d_{near}(i)$. This is given by $d_{near}(i) = d_{rnd}(i) + \alpha$.

4. Unmap $d_{near}(i)$ using a reverse lookup. The reverse lookup is performed using a hash table that relates constellation positions (e.g. $\frac{1}{\sqrt{10}} + j\frac{1}{\sqrt{10}}$) to bit combinations (e.g. 0000). This lookup provides a set of bits for each received modulated value $d(i)$.

In a single equation, finding the nearest value to a received modulated value, which is then demodulated, is given by

$$d_{near}(i) = \alpha + \left(\mathfrak{Re}\{2\alpha\} \lfloor \frac{\mathfrak{Re}\{d_{adj}(i)\}}{\mathfrak{Re}\{2\alpha\}} \rceil + j\mathfrak{Im}\{2\alpha\} \lfloor \frac{\mathfrak{Im}\{d_{adj}(i)\}}{\mathfrak{Im}\{2\alpha\}} \rceil \right). \tag{2.4}$$

While a reverse lookup is still required, the rounding method is much more computationally efficient because it doesn't have to calculate and compare distance from each constellation point. This method provides a constant time for all modulation schemes. Using this method reduces the number of required computations providing a much faster demodulation and overall evaluation.

Demodulation provides an output bitstream. Like before this bitstream can then be unpacked and turned back into data. While the end data is typically not very useful, the channel correction stage, the unmapping stage, and the output bitstream provide the capability to calculate the MSE, EVM, and BER respectively. These three metrics are commonly used to quantify the performance of algorithms used for OFDM wireless communications systems.

2.2.2 Analog Simulation

The analog simulation quantifies the effects of hardware components on the modulated and packetized data, and ultimately the EUT. The analog simulation also captures the impact of an entire phased array architecture along with wireless propagation channels. This section gives an in-depth description of the implementation of the analog simulation.

Software models represent the hardware components in the evaluator simulation. These models provide a black-box representation with well-defined inputs and outputs. This black-box representation allows for different representations of a hardware component (e.g., models for phase-shifters from two different vendors) without code changes. These models are defined either parametrically (i.e. through equations) or through scattering parameters (S-parameters) from measuring real components. The black-box nature of these models allows for easy simulation of a variety of phased array architectures. A base code that all models inherit from can be found in Listing B.1 and Listing B.2.

A set of equations and parameters defines parametric models. These models assume no reflections or back-propagation (e.g., $S_{11} = S_{22} = S_{12} = 0$). With these assumptions, the calculation of these models is given as

$$Y(f) = X(f)S(f), \tag{2.5}$$

where $Y(f)$ is the model output, $X(f)$ is the model input, and $S(f)$ is the frequency response of the model. Some models for variable amplifiers and phase shifters are state-defined, with responses dependent on the hardware state. For example a continuous phase shifter can be created by defining a parameter $shift$ as the desired phase shift and defining the hardware component response as

$$Y(f)_{PS} = e^{j(shift)} \tag{2.6}$$

where $shift$ is calculated for the frequency and phase shift that is desired (e.g. for beamsteering this would be defined as $shift = \frac{2\pi f}{c}\ \overline{k}_i\left(\theta_i, \phi_i\right) \cdot \overline{r}_n$). This continuous definition of a phase shifter can be extended to define a discretized phase shifter with N states by rounding the value of $state$ to the nearest multiple of $\frac{2\pi}{N}$. Examples of the implementations of a continuous and discretized phase shifter can be found in Listing B.3 and Listing B.4, respectively.

Scattering parameter (S-parameter) based models provide a closer representation of the hardware components and their uncertainties than parametric models. These models use measured S-parameters to represent the response of each component. S-Parameter uncertainties can be included as a mixture of noise distributions from the nominal response (e.g. a distribution derived from multiple measured responses), uncertainties from the measurements themselves, or as a random selection from a set of possible measured values. These

models have the limitations that (1) the models are valid only within the measured frequency range, and (2) any frequencies that were not measured but are within that range are only as accurate as the method used to estimate their response. An example implementation of a component defined by a single S-parameter measurement is given in Listing B.5. For components such as phase shifters that have multiple possible states (with a different S-parameter measurement for each state), this definition is extended providing the implementation in Listing B.6. Accurate simulation of multiple connected S-parameter components requires a composite S-parameter matrix. Following [63], this composite matrix is calculated by first transforming the S-parameters of each component to transfer parameters (T-parameters). These T-parameters are then multiplied to give a combined T-parameter response. This combined T-parameter response is then transformed back to S-parameters to provide a composite (cascaded) S-parameter matrix.

The phased array architecture simulation, a part of the framework, requires a scalable design. This scalability is achieved by allowing models to have a size and therefore be repeated a given number of times. This is achieved by defining multiple components of the same type (e.g. phase shifters) as a single python object. This object contains arrays to define important information of the phase shifters such as the current state they are in (for discrete phase shifters) or the shift they are set to (for continuous phase shifters). There is a corresponding entry in each array for all phase shifters. When simulating with the phase shifters, inputs to all of the phase shifters are then provided at once as an array and all simulation is performed in a highly vectorized manner. This method provides minimal memory overhead for each component (one entry per each array per each component) while also allowing for vectorized computations to take advantage of libraries like MKL [64] or graphics processing units (GPUs). This can also quickly create a phased array with any number of phased array elements. This simulation approach provides a robust and efficient simulation of large-scaled phased arrays of up to 10,000 antenna elements.

Once the components have been created with the desired sizes, they can be grouped together. The components are then into a single large component within the software. The components are compiled together to chain the output of one component to the input of the next component. Compilation also ensures S-parameter components are correctly cascaded. The output object then provides a simple interface to pass data through all components in the group. A partial view of the implementation of this component grouping can be seen in Listing B.7.

The phased array is simulated as a combination of antennas, amplifiers, phase shifters, and combiners. A visualization of this hardware component layout is given in Figure 2.7. The phased array beam is steered by changing the state of the phase shifter models to produce the necessary shift for each antenna element. These shifts are dependent on element location and the desired steering angle. The element positions of a phased array, r_n, the array steering direction θ_s, ϕ_s, and the angle of an incident plane wave θ_i, ϕ_i are shown

in Figure 2.8. It is important to note that θ_i, ϕ_i are always defined as the angle before the incident plane wave passes through the origin. For example, with an incident plane wave described by $\theta_i = \frac{\pi}{4}, \phi_i = 0$, the receive array will see the strongest signal at a steering angle of $\theta_s = \frac{\pi}{4}, \phi_s = 0$. These are defined using the same coordinate system as in section A.1.

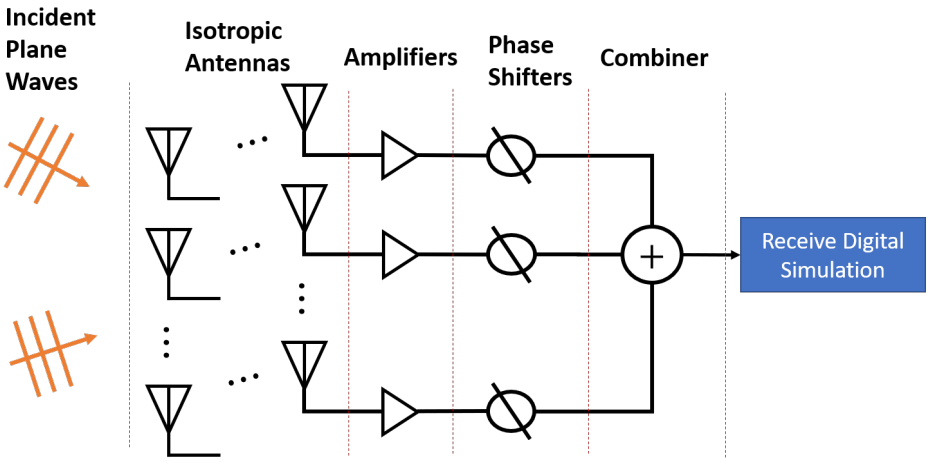


Figure 2.7 Visualization of a phased array simulated by the evaluator.

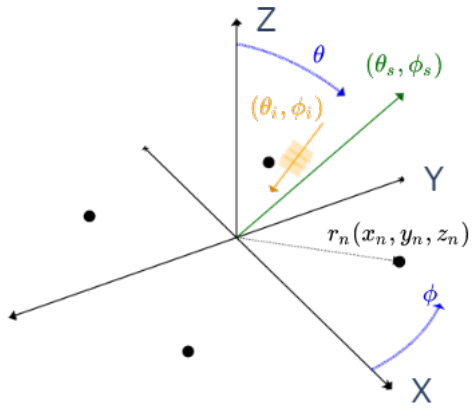


Figure 2.8 Coordinate system used for angle and position definitions. Examples of possible phased array element locations are given by the black dots. While this image shows a planar array ($z_n = 0$), these elements can exist at any location (x_n, y_n, z_n).

2.3 Millimeter-Wave Propagation Channels

Propagation channels are emulated using both synthetic and measured propagation channel models. Each type of channel model can be used interchangeably within the evaluator simulation. Synthetic channel models replicate ideal propagation channels with known incident angles, while measured channel models provide the ability to test more realistic environmental effects in the evaluator.

2.3.1 Synthetic Propagation Channels

Synthetic channel models are generated as a superposition of incident plane waves from a set of incident angles θ_i, ϕ_i (following the coordinate system defined in Figure A.1). These plane waves are a frequency domain version of the far-field approximation definition given in [65]. The received electric-field, E_n, at each element n for a set of I incident plane waves with $i = [1, ..., I]$ is given as

$$E_n(f) = \sum_{i=1}^{I} (\widehat{\theta} E_{i\theta}(f) + \widehat{\phi} E_{i\phi}(f)) e^{-j\frac{2\pi f}{c}\ \overline{k}_i(\theta_i,\phi_i)\cdot\overline{r}_n} \tag{2.7}$$

where $\widehat{\theta} E_{i\theta}$ and $\widehat{\phi} E_{i\phi}$ are the polarized frequency-dependent E-field magnitudes for each incident plane wave i. The phase shift $e^{-j\frac{2\pi f}{c}\ \overline{k}_i(\theta_i,\phi_i)\cdot\overline{r}_n}$ is dependent on the incident angle of each plane wave (θ_i, ϕ_i), the location of each receiving element, n, and frequency, f, with

$$\overline{r}_n = \widehat{x}x_n + \widehat{y}y_n + \widehat{z}z_n \tag{2.8}$$

and

$$\overline{k_i}\left(\theta_i, \phi_i\right) = \widehat{x}\sin\left(\theta_i\right)\cos\left(\phi_i\right) + \widehat{y}\sin\left(\theta_i\right)\sin\left(\phi_i\right) + \widehat{z}\cos\left(\theta_i\right). \tag{2.9}$$

Standardized frequency-dependent responses like those defined in the 3GPP specification for various environments [7] are used to generate $\widehat{\theta}E_\theta$ and $\widehat{\phi}E_\phi$ for each incident wave.

In the evaluator, each synthetic propagation channel is defined by a collection of these incident fields and their respective frequency dependent $\widehat{\theta}E_\theta$ and $\widehat{\phi}E_\phi$ values. The calculation of 2.7 resulting in the value of the received E-Field is actually calculated by the antenna models when using synthetic propagation channels. This allows antenna radiation patterns and polarizations to be accounted for in the evaluation by multiplying the antenna frequency response at each incident angle by the propagation channel frequency response. This is done for both polarizations of electric field to give an output voltage and current from the antenna. The output of the antenna is then provided as the input into the next hardware component (an amplifier in all cases presented).

2.3.2 Measured Propagation Channels

The evaluator is capable of directly integrating measured propagation channels to test how they compound with hardware and affect the efficacy of channel estimators. Each measured propagation channel was measured using the synthetic aperture measurements for uncertainties with angle of incidence (SAMURAI) system [66]. The goal of this system is to provide a flexible millimeter-wave channel measurement system while also providing measurement uncertainties from a variety of sources like calibrations and spatial uncertainty on top of these measurements. Measurements were made to characterize and bound the performance and uncertainty of the system [67]. The base components of the SAMURAI system are a 6-axis robotic arm for antenna positioning, and a vector network analyzer (VNA) for measurement. The system therefore measures the frequency domain scattering parameters which can be utilized during simulation.

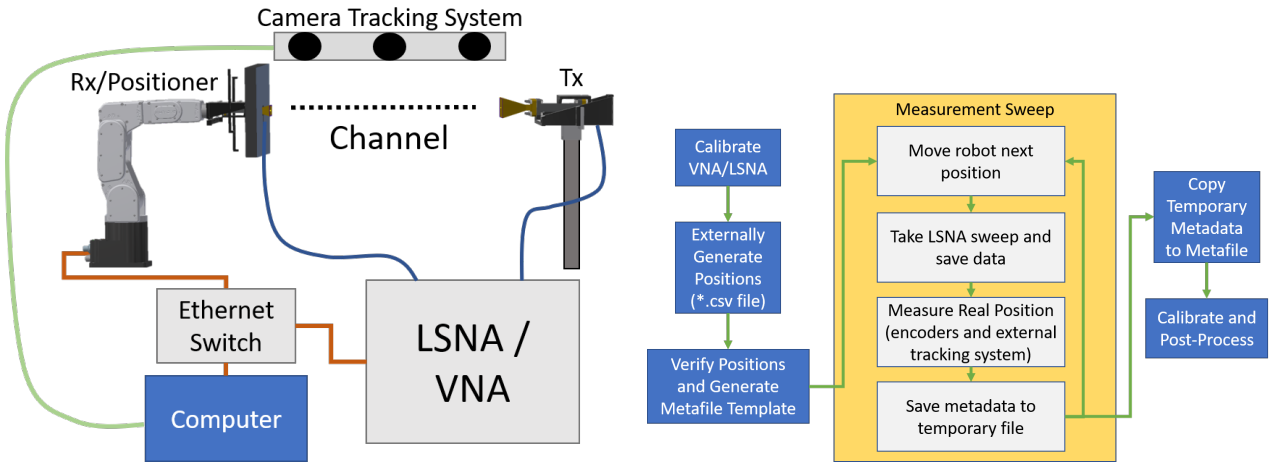
There currently exists a variety of different mmWave over-the-air (OTA) channel measurement systems based on different measurement techniques and hardware designs [68, 69, 70]. The SAMURAI system was designed specifically for flexible ultra wide band (UWB) measurements and is therefore based around a vector network analyzer (VNA) similar to other developed channel measurement systems [71, 72]. The usage of a VNA provides many advantages over other measurement tools is the ability to utilize the VNA in a non-linear large signal network analyzer (LSNA) setup. This then allows not only the measurement of linear scattering parameters, but other non-linear devices such as mixers and amplifiers. This capability is especially important for 5G millimeter-wave systems as the losses are very high at millimeter waves and

therefore devices are becoming packaged into a single device. For example a single chip may consist of everything from amplifiers and mixers, all the way to antennas. This creates difficulty when performing measurements of these systems as part of the measurement must connect with a cable and the other must be conducted over the air. The capability of the SAMURAI system to provide non-linear measurements was demonstrated in the measurement of up and down converting phased array systems [73]. Another important part of VNA/LSNAs for the SAMURAI system is the fact that traceable calibrations have previously been developed for VNA/LSNA hardware [74, 75, 76, 77]. These calibrations provide well proven techniques for calibration of VNA/LSNAs along with the propogation of uncertainties from these calibrations into the measurements performed by the device. The great measurement capabilities of the VNA come at a cost of measurement time. The VNA and LSNA both measure a single frequency at a time and sweep across all frequencies of interest. While this allows them to be very wideband in their measurements, they have a small instantaneous bandwidth. Wideband measurements therefore take much more time than they do in other systems. VNA based systems are typically restricted to measurements of slowly varying environments for this reason (all surroundings must stay static during the frequency sweep).

The second main piece of the SAMURAI system is a small 6-axis robotic arm. As previously mentioned, this system relies on a synthetic aperture method to be able to measure channels both as a function of frequency and angle. A synthetic aperture measurement works by physically placing a single antenna at multiple locations. This is in contrast to a phased array measurement system which has multiple elements which are coherently combined. An advantage of using a synthetic aperture method is that it can reduce the cost of a system by only requiring one antenna and a single receiver. This again though comes at the cost of measurement speed and is therefore restricted to only slowly varying channels. This is because the positioner must move, measure with the VNA, move to the next spot, measure, and continue until measurements have been taken at all positions. This does allow the added benefit though of all beam steering and angle of arrival estimation to be performed in post processing as opposed to in a typical phased array system which performs beam steering in hardware during measurement time. Compared to some of the other aforementioned measurement systems that utilize a synthetic aperture method, the inclusion of a 6-axis robot arm provides increased flexibility in the shapes of apertures that can be created. For example, systems like that presented in [71] utilize multiple linear slide stages to achieve 2D and 3D aperture capabilities. 6-axis robotic arms increase the number of positions at which the antenna can be placed by providing both 3D placement of the antennas along with rotation of the antenna and shifting of the polarization. Each of these different positions can be utilized to create arbitrary apertures with arbitrary polarizations and antenna rotations with a single measurement system. This type of positioning system has successfully been implemented for OTA measurements in the past in other measurement applications like near field antenna

measurements [78].

These two major parts are then both connected to a computer for orchestration of all of the measurements. An optical based position tracking system is also included in the system. This provides external tracking of both the position and orientation of the synthetic aperture antenna along with any scatterers in the channel and the antenna that is transmitting to the synthetic aperture. This tracking also allows verification of algorithms for things such as angle of arrival by providing a good estimate of the geometry of the channel being measured which can be used as a ground truth for verification. Typically measurements are performed on a large optical table providing repeatable placing of scatterers. A system diagram and operational flow chart can be seen in Figure 2.9 with a line-of-sight measurement setup on an optical table seen in Figure 2.10.



(a) Diagram of SAMURAI hardware.

(b) Flow chart of SAMURAI operation.

Figure 2.9 Samurai system overall configuration.

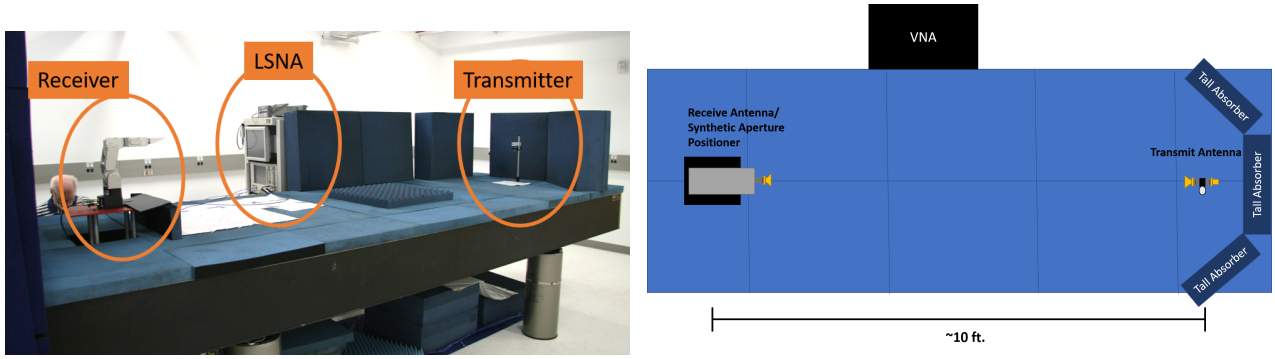


Figure 2.10 Images depicting the setup of the SAMURAI system for measuring a line-of-sight configuration on an optical table covered with absorbing material.

Because of the usage of a VNA, the frequency spacing and bandwidth that the SAMURAI system can measure is extremely flexible. In order to directly adhere to the 5G millimeter-wave standard, a variety of measurements were taken with bandwidths in the 27.5 GHz and 39 GHz frequency bands in a variety of environments. To allow for direct integration with an OFDM simulation, each of these measurements was set to measure the correct subcarrier spacing for 5G systems as defined by 3GPP. The frequency spacing of the VNA for each measurement for OFDM simulation was therefore set to 15 $kHz * 2^n$ for $n = 1, 2, 3, 4$. More information on the 5G frequencies and subcarrier spacing can be found in section 1.2. The total measurement time depends on a variety of factors on the VNA and synthetic aperture setup, but the main factors are number of frequency points to measure and number of aperture positions that are being measured. For this reason, larger apertures were typically measured with smaller bandwidths or larger subcarrier spacing compared to measurements taken with smaller apertures.

SAMURAI measurements are calibrated and calibration uncertainties are captured through typical VNA/LSNA calibration routines. The calibrations are performed by taking un-calibrated measurements of calibration standards comprised of high precision coaxial or waveguide components that have been characterized during fabrication to provide known responses. While many calibration methods exist, this system utilized a short-open-load-through (SOLT) configuration. Measurements are then calibrated using this data in post-processing. Calibration in post-processing provides the capability to add uncertainties from the calibration while also allowing flexibility with testing multiple calibration routines without needing to re-measure the data. After the measurement of initial calibration standards, aperture positions are externally generated as a comma separated value (CSV) file. Each line in this file is a position and orientation of the robot to measure in the synthetic aperture. At each of the positions, the SAMURAI system will measure a full VNA/LSNA sweep that is set up before the measurement. At each position, data from the optical tracking system along with any other desired data is also stored in a temporary file. In order to adhere to a standard file format, all sweeps measured with the VNA are stored in a touchstone (*.snp) file format. This allows interoperation with a variety of programs and tools. After a measurement has been performed at each predefined position, all data stored during the measurement is formatted into a human-readable metafile that both contains a variety of information on the measurement along with the paths to all of the measured VNA sweeps. The VNA/LSNA post-calibration is then performed to capture any drift in the system during its measurement period. The data from the SAMURAI sweep is all calibrated in software post-processing using the measured responses of the SOLT standards. Capturing the drift is important for this type of system as measurements may take a day or more to complete. All measurements are calibrated using the previously developed NIST microwave uncertainty framework (MUF) [79]. Calibration with the MUF corrects the measurements to a known reference plane using a SOLT calibration algorithm [76]. The calibration also pro-

vides uncertainties on the measurements from the drift in the system using the differences between the pre- and post-calibrations. Once the data has been calibrated, it can directly be used in the channel estimator evaluator. This is because the simulation software is built to utilize frequency domain measurements. For evaluation at frequencies that were not measured, linear interpolation and nearest neighbor extrapolation are used to estimate the response at the unmeasured frequencies.

The calibrated data at the frequencies of interest can then be integrated into the evaluator. Unlike synthetic propagation channels, the measured propagation channels consist of a frequency response at each antenna element in the system. Therefore, the evaluator currently is built with the assumption that the antenna model is a non-polarized isotropic antenna. Any antenna gain or polarization adjustments must be made to the data before integration into the evaluator. Due to the current way the evaluator is designed, integration of measured propagation channels assumes that the frequency response data for each element is already a set of voltages and currents. The output of the measured propagation channels, is therefore provided directly to the input of the next hardware component.

2.4 Monte Carlo Analysis

Monte Carlo analysis is used in the evaluator to capture the effects of any hardware component noise and systematic offsets on the channel estimators. These iterations consist of two parts. First, the iterations are repeated simulations of multiple OFDM symbols. Repeating multiple symbols provides variations in the pilot tones within a single estimation. Each transmitted symbol captures different effects introduced by the uncertainties of the hardware components. This variation may change the efficacy of an estimator, for example, if the estimator uses both frequency (subcarriers) and time (symbols) data when performing estimation. This for example could be used in a 2D interpolation scheme to estimate the channel response between each pilot tone. Second, the evaluation performs repeat evaluations to ensure that any variations in the evaluation due to the uncertainties in hardware components are captured. This allows the generation of a noise distribution for the evaluation of each estimator. Further testing and usage of these repeat evaluations is given in section 2.10.

2.4.1 System-Level Metrics

Once the evaluator has completed the Monte Carlo analysis from section 2.4, system-level metrics are generated: MSE, EVM, and BER, to quantify the efficacy of the EUT. The MSE is calculated as

$$\mathrm{MSE}\left(H_{\mathrm{est}}\right) = \frac{\sum_{n=0}^{N} |H - H_{\mathrm{est}}|^2}{N} \tag{2.10}$$

where H_{est} is the estimated response of the channel for each subcarrier and symbol, H is the actual response of the channel, and N is the total number of subcarriers times the number of symbols. This metric provides

a measure of how much error is in the actual estimation of the channel. A perfect estimator for a wireless communication system without any error produces a value of zero. The MSE value rises with increasing uncertainties and imperfect estimations.

The system-level metric EVM is calculated similarly to [59, 60] as

$$\mathrm{EVM}\left(D_{\mathrm{rx}}\right) = 100 \times \frac{\sqrt{\sum_{n=0}^{N} |D - D_{\mathrm{rx}}|^2}}{\sum_{n=0}^{N} |D|^2} \tag{2.11}$$

where D are the correct modulation constellation values and D_{rx} are the received modulation constellation values after channel correction, giving an output as a percentage. Like the MSE, EVM provides a normalized error estimation but is one step removed directly from the channel estimation itself. This metric shows how a change in the MSE of the estimated channel can affect the received data for a given modulation scheme.

Finally, calculating BER provides a metric that is one more step removed from the EVM calculation. The BER can be calculated as

$$\mathrm{BER}\left(N_{\mathrm{err}}\right) = \frac{N_{\mathrm{err}}}{N_{\mathrm{bits}}} \tag{2.12}$$

where N_{err} is the number of incorrect bits, and N_{bits} is the total number of bits. As opposed to bit error rate, this provides a unitless metric on how many bits are affected in a given amount of data without a time dependency.

Each of these system-level metrics is important as they all provide insight into how well an EUT affects a different part of a wireless communication system.

2.5 Verification of Evaluator

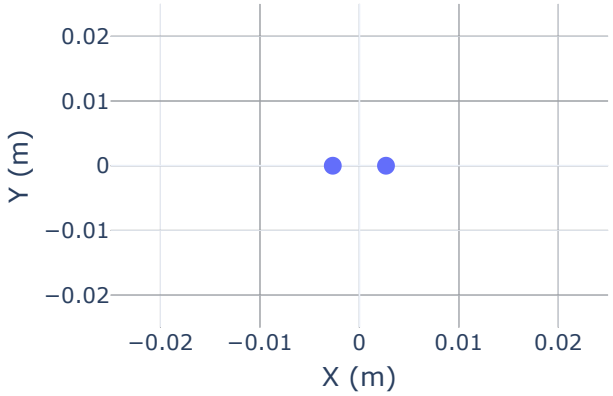
In this section, extensive verification tests of the channel estimator evaluator performance and accuracy are detailed. Implementation errors during the development of the evaluator can occur in the models, uncertainties, or the simulation itself, leading to inaccurate testing of the channel estimators. The verification process followed published guidelines that detail effective ways to ensure the validity of results from electromagnetic software [80]. Further, the verification approach discussed here can apply as a general guide for verifying a wide variety of custom wireless communication software.

2.5.1 Estimator Evaluator Verification Configuration

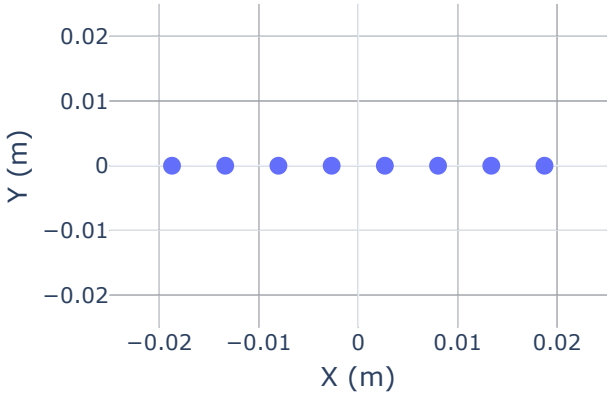
For verification, transmit side of the evaluator simulation is simplified to be a superposition of plane waves. Therefore, the transmit side is assumed to have ideal hardware and resides at the far-field of the receiver to produce plane waves. This simplification focuses the investigation on the digital and receiver analog hardware in the evaluator simulation. The transmitter analog hardware simulation is equivalent to the receiver for verification. It is also assumed that there is no loss due to polarization mismatch. The expression

of the electric-fields for incident plane waves is therefore simplified such that $E_i(f) = \left(\widehat{\theta}E_\theta(f) + \widehat{\phi}E_\phi(f)\right)$. The effect on planar arrays is being evaluated so $z_n = 0$ for each $\overline{r_n}$. An analog receive phased array architecture comprised of an amplifier followed by a phase shifter behind isotropic antenna elements was used. The output of each phase shifter is then combined before being passed to the receive digital section of the evaluator simulation [81*]. These phased array hardware components are defined by and tested with several different hardware component models ranging from ideal parametric models to measured S-parameter models. These hardware components are laid out according to Figure 2.7.

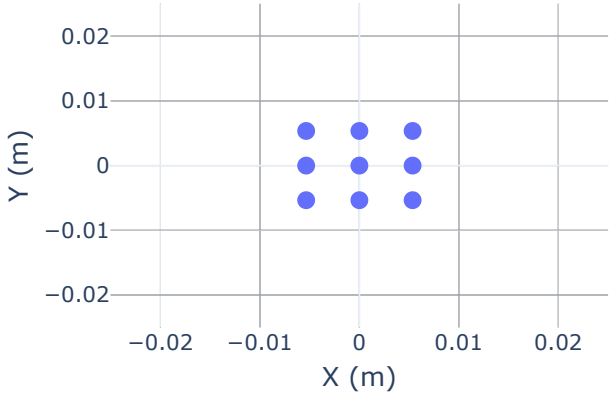
For verification, a 2-element uniform linear array (ULA) (Figure 2.11(a)), an 8-element ULA (Figure 2.11(b)), a 3x3 planar array (Figure 2.11(c)), and an 8x8 planar array (Figure 2.11(d)), all with 5.35 mm element spacing were tested operating at 27.5 GHz. These phased arrays were selected based on the requirements and scalability of each verification test.



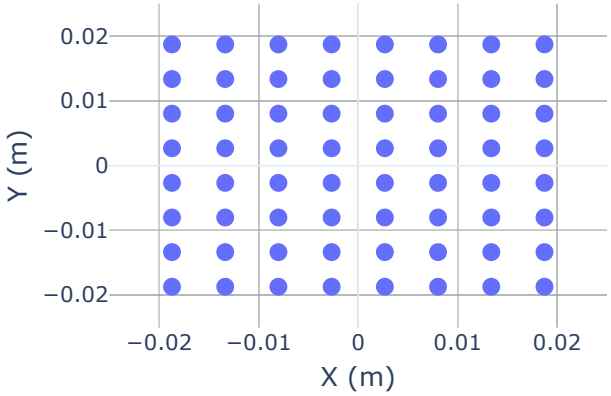
(a) 2-element uniform linear array.



(b) 8-element uniform linear array.



(c) 3x3 element planar array.

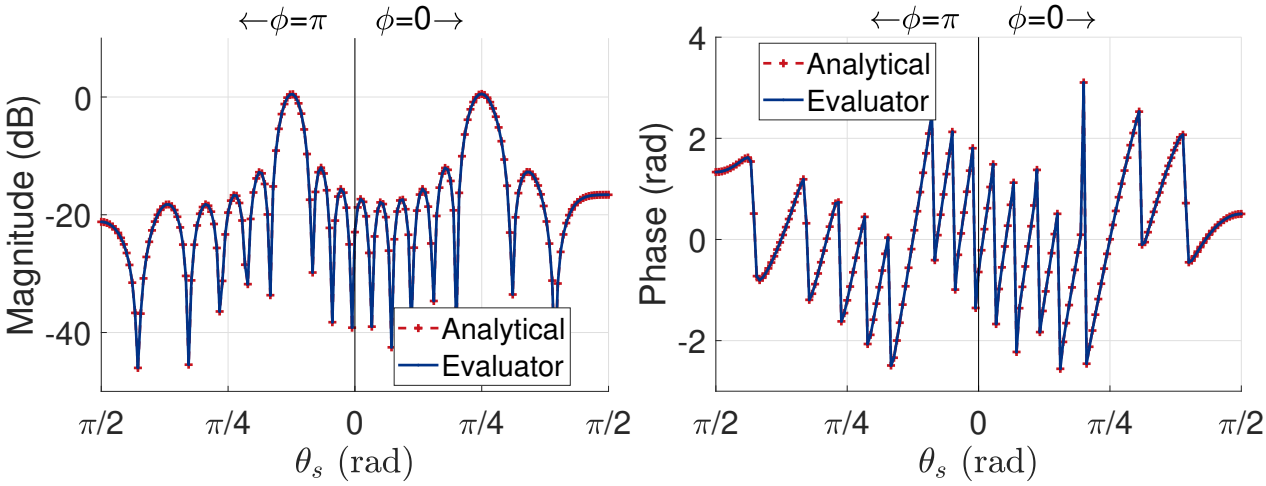


(d) 8x8 element planar array.

Figure 2.11 Layout of phased array elements used in testing.

2.5.2 Nominal Analytical Verification of System-Level Metrics

The evaluator's performance against an analytical beamforming solution was verified with a plane wave test channel. The verification process uses the evaluator with "ideal" hardware models that have no loss, error, or phase discretization. This evaluator configuration was compared against a software implementation of the analytical beamforming equation (see (2.13)) [66]. Figure 2.12 shows minimal mismatch (7×10^{-13}) dB maximum) between the analytical beamforming software (Analytical) and the evaluator with "ideal" models (Evaluator) for a propagation channel with known incident angles.



(a) Magnitude

(b) Phase

Figure 2.12 Spatial responses of an analytical beamforming solution and the evaluator with ideal hardware models (continuous phase shifters, unity gain amplifiers, ideal combiners as seen in Figure 2.7) for known incident angles.

Modulation and demodulation of the evaluator was tested using both simulated and measured channels alongside noise-free hardware models. For a least squares channel estimator, the MSE, BER, and EVM equal zero (omitting machine error), which agrees with the analytical solution for a static channel with no error. A result of zero is expected for this test case.

2.5.3 Analytical Verification of Monte Carlo Simulations

A step in verification was to compare the output of the evaluator's Monte Carlo simulations against an analytical solution. The comparison was performed with a 2-element ULA with parametric models for unity gain amplifiers, continuous phase shifters, and an ideal combiner comprised of the receiver. A small array was used due to the complexity and solving time of the analytical solution. All noise was sampled from zero-mean Gaussian distributions. The distribution for phase noise had a 5-degree standard deviation,

while magnitude noise had a 0.001 linear standard deviation. These distributions were used to simplify the calculation of the expected value.

The expected value for the phased array can be calculated as the expected value of a multi-variate distribution. This derivation starts with the expression for conventional beamforming

$$F\left(f,\theta_s,\phi_s\right)=\frac{1}{N}\sum_{n=1}^{N}E_n\left(f\right)W_n e^{j\frac{2\pi f}{c}\ \overline{k_s}(\theta_s,\phi_s)\cdot\overline{r}_n} \tag{2.13}$$

where the output $F\left(f,\theta_s,\phi_s\right)$ is given by the coherent combination of the received electric-field at each antenna element $E_n\left(f\right)$, the weighting at each antenna element, W_n. The steering vector for the angle of interest is defined by $e^{j\frac{2\pi f}{c}\ \overline{k_s}\cdot\overline{r}_n}$ where $\overline{r}_n$ is given by (2.8) and $\overline{k_s}=\overline{k_i}\left(\theta_s,\phi_s\right)$ is given by (2.9) where θ_s,ϕ_s are the steering angles of the phased array (i.e., boresight is at $\theta_s=0,\phi_s=0$). The values of E_n and W_n are then replaced with the channel response at each element (h_{OTA_n}) and the frequency response of component m behind element n (HW_{mn}) with M total components behind N total elements. Replacing these values then yields

$$F(f,\theta_s,\phi_s)=\frac{1}{N}\sum_{n=1}^{N}h_{\mathrm{OTA}_n}\left(\prod_{m=1}^{M}HW_{\mathrm{mn}}\right)e^{j\frac{2\pi f}{c}\overline{k_s}\cdot\overline{r_n}}. \tag{2.14}$$

The array shift from the phase shifters are not included in HW_{mn} but in the steering vector, although any other response components (e.g., loss, noise) are in HW_{mn}. Equation (2.14) is then extended to include our uncertainties on each of our hardware components to get the value F with added uncertainties as F_δ. This is defined as

$$F_\delta(f,\theta_s,\phi_s)=\frac{1}{N}\sum_{n=1}^{N}h_{\mathrm{OTA}_n}\left(\prod_{m=1}^{M}\left(HW_{\mathrm{mn}}+\delta_{\mathrm{mn}}\right)\right)e^{j\frac{2\pi f}{c}\overline{k_s}\cdot\overline{r_n}}, \tag{2.15}$$

where δ_{mn} is a random complex variable for the noise from component, m, behind the element, n. For simplicity, the following is also defined

$$A_{\delta\mathrm{n}}(f)=|h_{\mathrm{OTA}_n}|\left(\prod_{m=1}^{M}\left(|HW_{\mathrm{mn}}|+|\delta_{\mathrm{mn}}|\right)\right) \tag{2.16}$$

$$C_{\delta\mathrm{n}(f)}=\frac{2\pi f}{c}\overline{k}_s\cdot\overline{r}_n+\arg\left(h_{\mathrm{OTA}_n}\right)+\sum_{m=1}^{M}\left(\arg\left(HW_{\mathrm{mn}}\right)+\arg\left(\delta_{\mathrm{mn}}\right)\right). \tag{2.17}$$

By separating the real and imaginary components, the expected value of $F_\delta(f,\theta_s,\phi_s)$ ($E[F_\delta]$) is then calculated [82]. This separation gives

$$
\begin{aligned}
&E\left[\mathrm{Re}\left\{F_\delta\right\}\right] = \\
&E\left[\sum_{n=1}^{N} A_{\delta \mathrm{n}} \cos\left(C_n + \sum_{m=1}^{M} \arg(\delta_{\mathrm{mn}})\right)\right] = \\
&\int \cdots \int_{-\infty}^{\infty} \left(\prod_{m=1}^{M} P_{\mathrm{mn}}\left(\delta_{\mathrm{mn}}\right)\right) \left[\sum_{n=1}^{N} A_{\delta \mathrm{n}} \cos\left(C_{\delta \mathrm{n}} + \sum_{m=1}^{M} \arg(\delta_{\mathrm{mn}})\right)\right] \partial\delta_{11} \cdots \partial\delta_{\mathrm{MN}}
\end{aligned}
\tag{2.18}
$$

$$
\begin{aligned}
&E\left[\mathrm{Im}\left\{F_\delta\right\}\right] = \\
&E\left[\sum_{n=1}^{N} A_{\delta \mathrm{n}} \cos\left(C_n + \sum_{m=1}^{M} \arg(\delta_{\mathrm{mn}})\right)\right] = \\
&\int \cdots \int_{-\infty}^{\infty} \left(\prod_{m=1}^{M} P_{\mathrm{mn}}\left(\delta_{\mathrm{mn}}\right)\right) \left[\sum_{n=1}^{N} A_{\delta \mathrm{n}} \sin\left(C_{\delta \mathrm{n}} + \sum_{m=1}^{M} \arg(\delta_{\mathrm{mn}})\right)\right] \delta_{11} \cdots \partial\delta_{\mathrm{MN}}
\end{aligned}
\tag{2.19}
$$

where P_{mn} is the probability density function (PDF) of component m behind element n. For this verification, magnitude and phase noise are assumed to be independent. Each PDF, P_{mn}, is then split into a product of PDFs for the magnitude and phase giving

$$
P_{\mathrm{mn}}\left(\delta_{\mathrm{mn}}\right) = P_{mn_{\mathrm{abs}}}\left(\delta_{\mathrm{mn}}\right) P_{\mathrm{mn}\arg}\left(\delta_{\mathrm{mn}}\right) \tag{2.20}
$$

which then gives

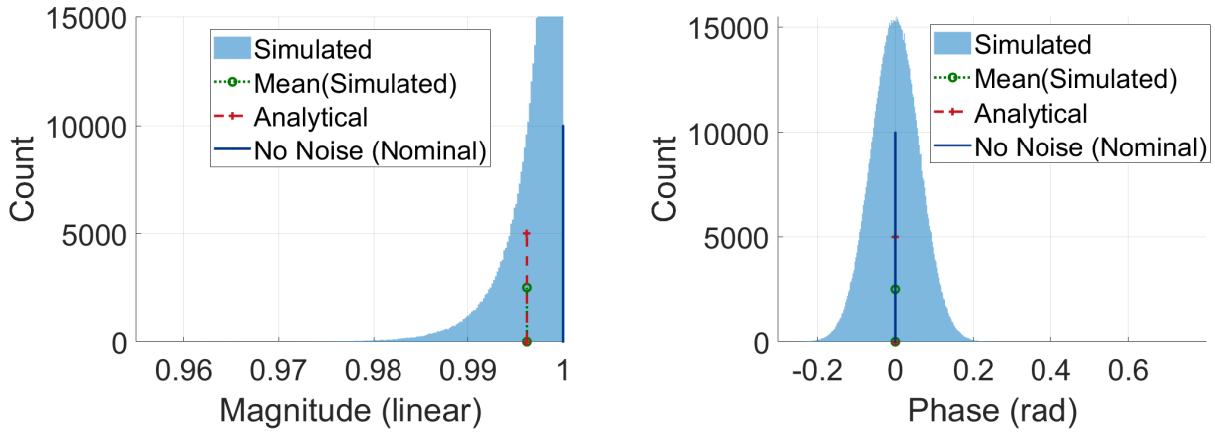
$$
\begin{aligned}
&E\left[\mathrm{Re}\left\{F_\delta\right\}\right] = \\
&\int \cdots \int_{-\infty}^{\infty} \prod_{m=1}^{M} \left(P_{mn_{\mathrm{abs}}}\left(\delta_{\mathrm{mn}}\right) P_{\mathrm{mn}\arg}\left(\delta_{\mathrm{mn}}\right)\right) \\
&\qquad \left[\sum_{n=1}^{N} A_{\delta \mathrm{n}} \cos\left(C_{\delta \mathrm{n}} + \sum_{m=1}^{M} \arg\left(\delta_{\mathrm{mn}}\right)\right)\right] \partial(arg(\delta_{11}))\ \partial\left(\mathrm{abs}\left(\delta_{11}\right)\right) \cdots\ \partial(arg(\delta_{\mathrm{MN}}))\partial(abs\left(\delta_{\mathrm{MN}}\right)
\end{aligned}
\tag{2.21}
$$

$$
\begin{aligned}
&E\left[\mathrm{Im}\left\{F_\delta\right\}\right] = \\
&\int \cdots \int_{-\infty}^{\infty} \prod_{m=1}^{M} \left(P_{mn_{\mathrm{abs}}}\left(\delta_{\mathrm{mn}}\right) P_{\mathrm{mn}\arg}\left(\delta_{\mathrm{mn}}\right)\right) \\
&\qquad \left[\sum_{n=1}^{N} A_{\delta \mathrm{n}} \sin\left(C_{\delta \mathrm{n}} + \sum_{m=1}^{M} \arg\left(\delta_{\mathrm{mn}}\right)\right)\right] \partial(arg(\delta_{11}))\ \partial\left(\mathrm{abs}\left(\delta_{11}\right)\right) \cdots \partial(arg(\delta_{\mathrm{MN}}))\partial(abs\left(\delta_{\mathrm{MN}}\right).
\end{aligned}
\tag{2.22}
$$

The analytical solution given by (2.21) and (2.22), was computed numerically using the Scipy nquad function in Python [83]. Each solution requires $2 \times M \times N$ integrations, limiting the size and complexity of the array that can be verified.

The analytical results were compared to the output of the evaluator's hardware component simulation. Expected values were calculated and compared for a 2-element array at a single steering direction with an

incidence plane wave of a magnitude of 1 incident at $\theta_i = 0, \phi_i = 0$ (boresight). These analytical and evaluator results were calculated separately for magnitude (shown in Figure 2.13(a)) and phase error (shown in Figure 2.13(b)) to reduce integration complexity and computation time. These plots also include the nominal result without any noise as a point of reference. It should be noticed here that the addition of phase noise causes a skew in the output distribution due to the incoherent combination of noise in the array.



(a) Magnitude

(b) Phase

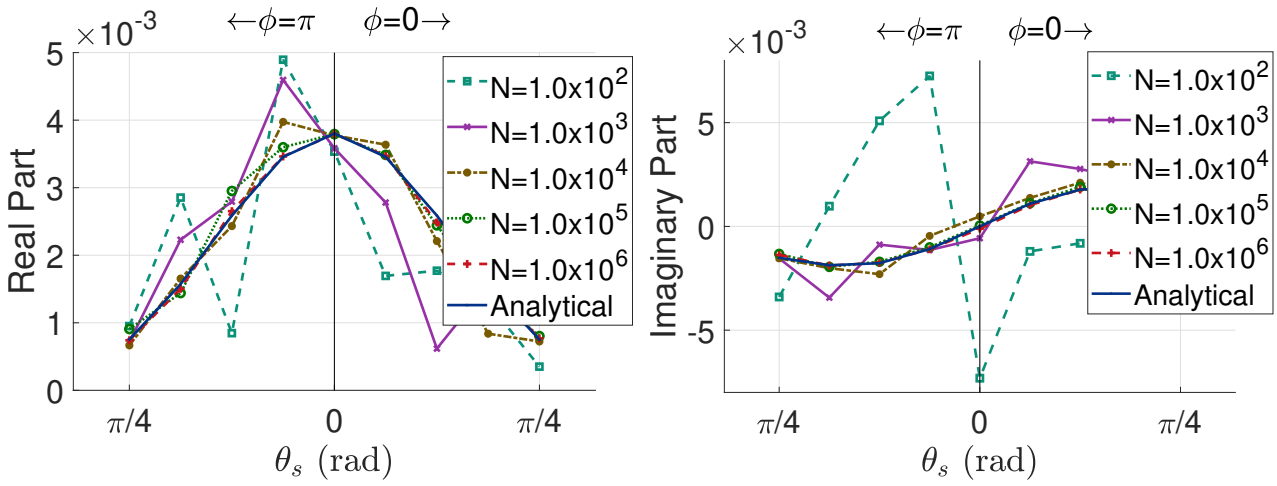
Figure 2.13 Expected vs. simulated phase value for added phase noise. Nominal (without noise) included as a reference. Expected and simulated are nearly identical to the nominal. Calculated as the angle of the complex number with the real part given by (2.21) and the imaginary part given by (2.22).

Table 2.3 Results of the analytical verification with noise.

Solver	**Magnitude Noise**	**Phase Noise**
Nominal	$1 + 0j$	$1 + 0j$
Analytical	$1.00 + 0j$	$0.996 + 0j$
This Evaluator	$1.00 + 0j$	$0.996 - 5.95 \times 10^{-}6j$

Table 2.3 contains the results of the analytical solution and the evaluator simulation, where the nominal result was taken as the polar mean of all the Monte Carlo results. While these results are similar, they would only be expected to match exactly as the number of Monte Carlo iterations approaches infinity. These simulations were repeated 1×10^6 times to get the approximate matching result seen here. The convergence as the number of Monte Carlo runs increases, as seen in Figure 2.14

A number of limitations were found when performing the analytical verification with noise. All of these limitations stemmed from the inability to solve the complex analytical equations required to represent the array and its hardware. As previously mentioned, the analytical solution requires $2 \times M \times N$ integrals for



(a) Real Part

(b) Imaginary Part

Figure 2.14 Spatial response of analytical solution vs. evaluator with a different number of Monte Carlo iterations. The evaluator result converges to the analytical result as the number of Monte Carlo iterations increases.

each array layout. While theoretically the analytical solution can be derived for any sized array, solving the solution becomes difficult as M and N increase. A number of methods were attempted to perform the high dimensional integration in the analytical solution. Symbolic solving was first used in an attempt to get an exact solution to the problem. No previously solved solution was available through sources such as [84]. Symbolic solving was then attempted using the python based symbolic solver SymPy [85]. This solver was unable to reach a solution for more than a single element with a single hardware component ($M = 1$, $N = 1$) which is not useful for the case of phased arrays (as 1 element is not an array). From there, numerical methods were tested in an attempt to get close to the correct value of the integral. This included attempting to estimate higher order integrals using Monte Carlo integration with packages like [86], but for low dimensions these did not produce accurate enough results when compared to typical numerical integration. Finally, the compromise was made to use a typical numerical integration package, in this case being the Scipy nquad function. While the usage of this numerical integration package limited the size of array and number of hardware components that could be simulated, it did allow the simulation of multiple antennas ($N > 1$) with at least 1 hardware component ($M > 0$) to provide a verification of the evaluator simulation.

2.6 Verification against Commercial Software

Next, phased array configurations with greater complexity were used to verify the evaluator performance by comparing against commercial software. This comparison allowed for the verification of more complex

components, such as those defined by S-parameters and phased array architectures with more antenna elements. This more complex verification was not possible when comparing against the analytical solution with noise because of the computational difficulty of solving the analytical solution. Further comparisons and verification were required to verify the evaluator in more complex environments. Tests were first run without noise to verify the nominal result of the channel estimator evaluator. Tests with noise were then run to generate comparable distributions. The test without noise ensure the evaluator is producing the correct output values while the tests with noise ensure it is producing the correct distribution. Each of these tests was conducted using Keysight's Advanced Design System (ADS) [41]. While commercial software provides a good tool for verifying the evaluator, the estimator evaluator provides a more cost-effective, scalable (for usage with very large arrays), and extensible (ease of adding new features) evaluation of channel estimator performance, including hardware uncertainties. For example, one limitation of ADS that was found during verification was that when using S-Parameter based components in a phased array, there is no way to programatically change states. This meant there was no way to steer an S-Parameter based array without manually setting S-Parameters for all phase shifters in an array. This makes scaling to large S-Parameter based arrays in ADS unrealistic whereas the evaluator is capable of scaling this sort of array configuration.

2.6.1 Verification against Commercial Software without Noise

Noise-free simulations of phased arrays were run in both ADS and the developed evaluator. The 8-element ULA and 3x3 element planar array were used in these verification comparisons. Each phased array was tested using the generic models used in the analytical verification. Hardware models based on measured S-parameters were then used to verify a more complex configuration. In ADS, the incident plane wave was simulated in ADS by a phase shifter attached to the input of each antenna element. The outputs of the phase shifters in this incident plane wave generator were set to match the expected value at each antenna element based on (2.7). By assuming perfectly isotropic antennas and no polarization mismatch, we can simplify the problem and pass the output of these plane wave generating phase shifters directly to the input amplifiers of each element in the receiver phased array. The setup in ADS for an 8 element ULA can be seen in Figure 2.15.

The 8-element linear array was swept from $\theta_s = \frac{\pi}{2}$ to $\theta_s = 0$ with $\phi_s = \pi$ and $\theta_s = 0$ to $\theta_s = \frac{\pi}{2}$ with $\phi_s = 0$ with a plane incident wave at $\theta_i = \frac{\pi}{8}, \phi_i = \pi$. These tests for a phased array with generic continuous phase shifters and unity gain amplifiers can be seen in Figure 2.16. The same generic array using discrete 5-bit phase shifters can be seen in Figure 2.17. The comparison for an array with S-parameter based models can be seen in Figure 2.18. Figure 2.16 and Figure 2.17 use equations to define their phase shifters and unity gain amplifiers while Figure 2.18 defines the amplifiers and phase shifters by measured S-parameter values of

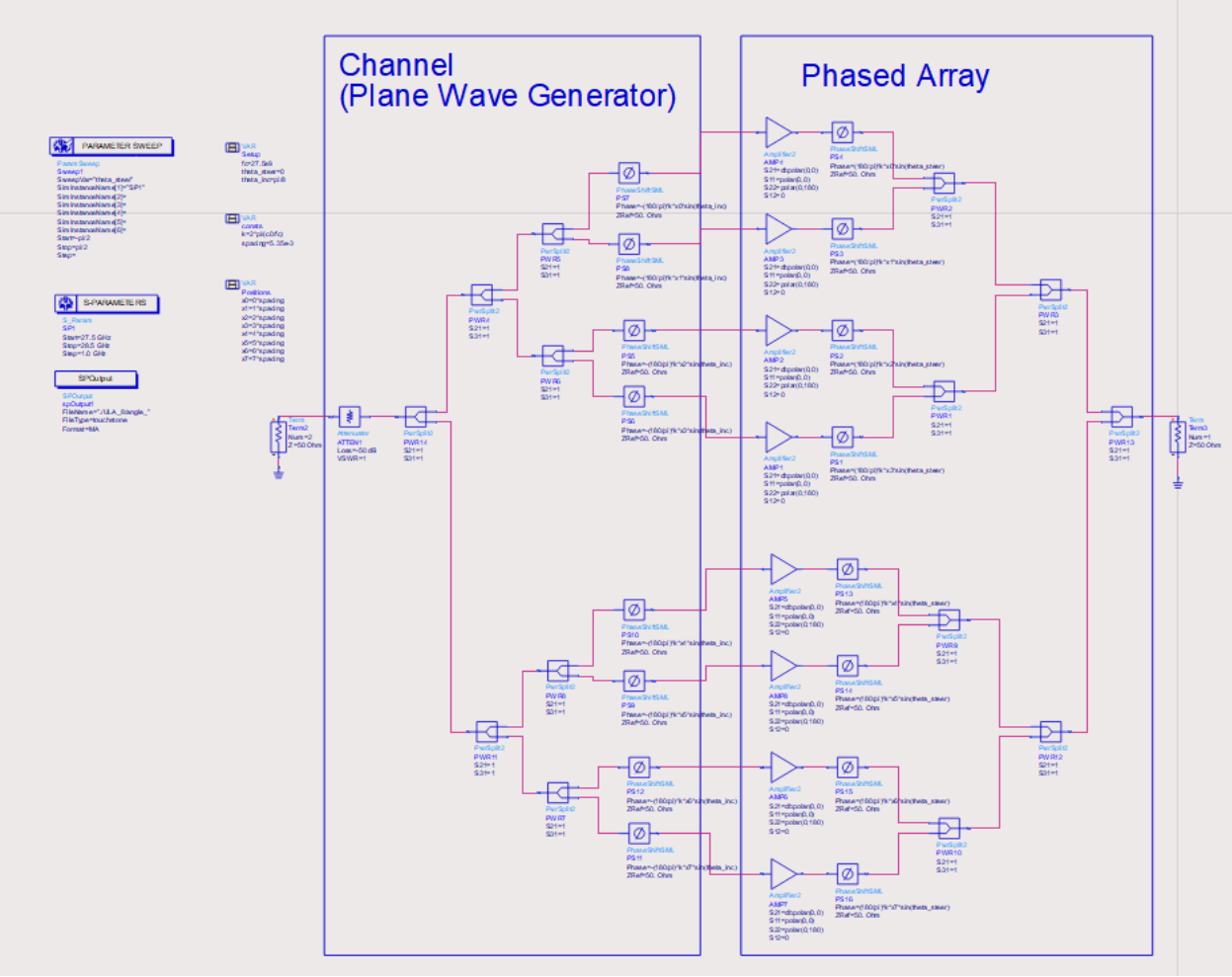
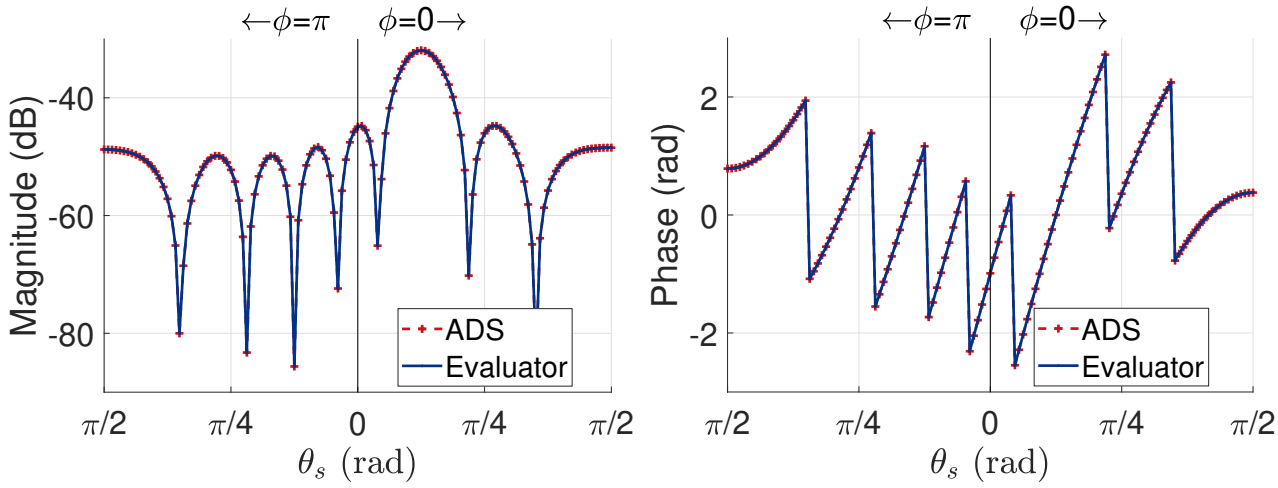


Figure 2.15 ADS schematic for verification of evaluator using an 8-element ULA.

a real phase shifter and amplifier. The spatial sweep for the parametric models and the S-parameter models from the evaluator match up with their ADS counterparts. This equality of the commercial comparison results further confidence that simulations using both parametric and S-parameter models work as expected.



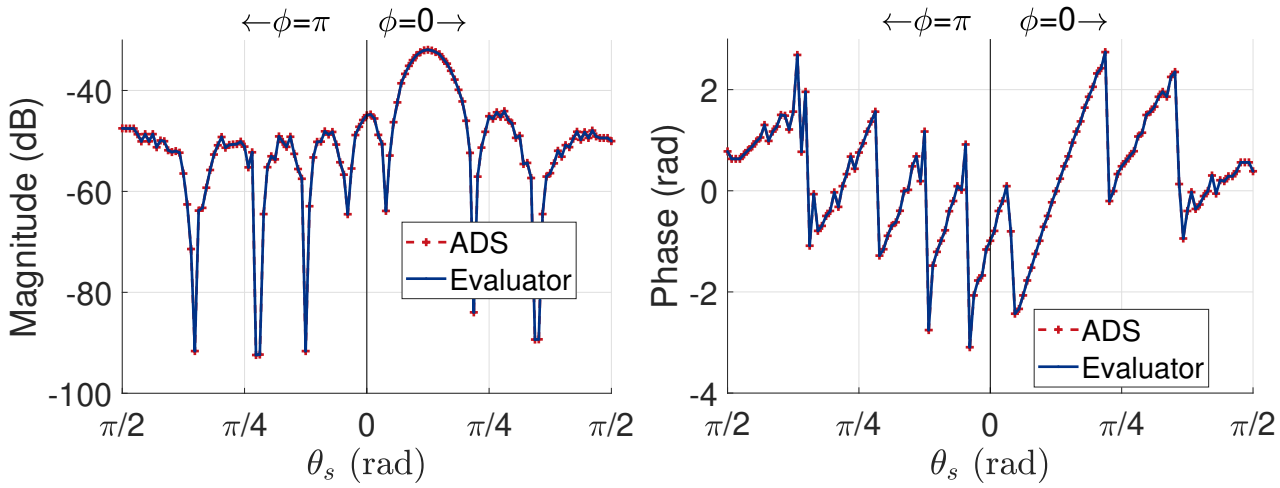
(a) Magnitude

(b) Phase

Figure 2.16 Spatial responses for simulations performed using the evaluator simulation and Keysight ADS with continuous phase shifters and unity gain amplifiers in an 8-element uniform linear array.

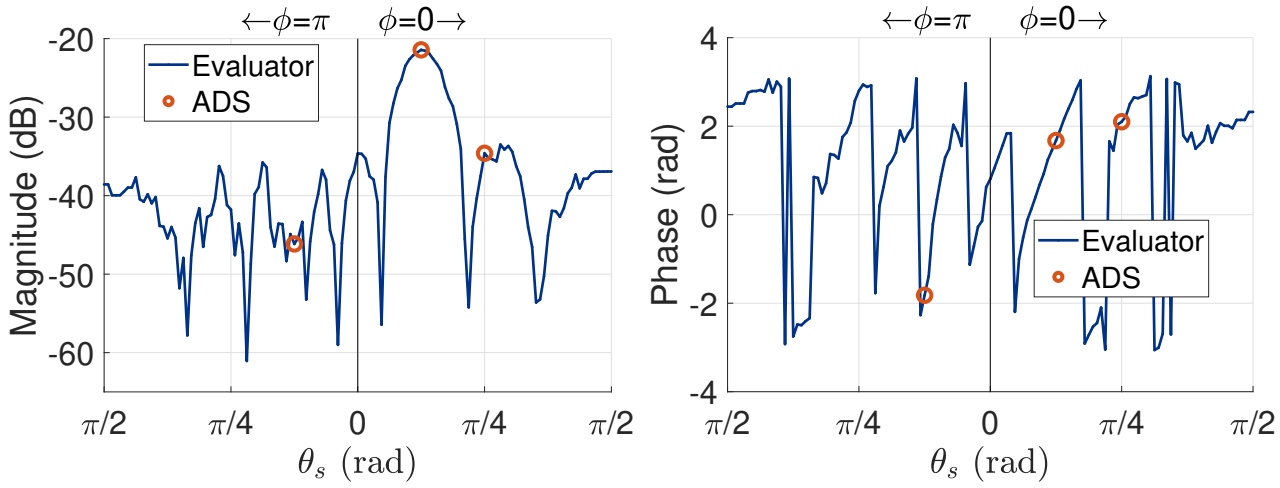
The 3x3 planar array was also simulated with a single incident wave at $\theta_i = \frac{\pi}{8}, \phi_i = \frac{\pi}{8}$ and the same S-parameter components used to generate Figure 2.18. The array was swept and compared across multiple different steering angles. Three steering angles were tested for $\phi_s = 0$ and $\phi_s = \frac{\pi}{2}$ the array S-Parameters as the correct phase shift response files had to again be selected manually. The results from the evaluator and ADS again matched all tested steering angles as seen in Figure 2.19. This verified functionality of the estimator evaluator simulation when $\phi_s \neq 0$.

By first verifying the calculation of the nominal response, issues with the evaluator's simulation of components without noise could be found. Specifically, this verification helped highlight an error that was occurring when cascading S-parameters. This verification step highlighted this issue, allowing it to be corrected. A pitfall of the verification method itself was also found. This pitfall existed in the comparison of the spatial sweeps due to the discretization of the sweep itself. Across the spatial sweep range, the values of θ and ϕ at which the responses were being calculated were slightly offset. These offsets were so small, they were not noticeable without extending the number of decimal places in the displayed results. This small offset was enough to cause the spatial responses to not match, especially at values of θ and ϕ far from the angle of incidence. This issue highlighted both the importance and difficulty of creating identical simulations between two different software packages.



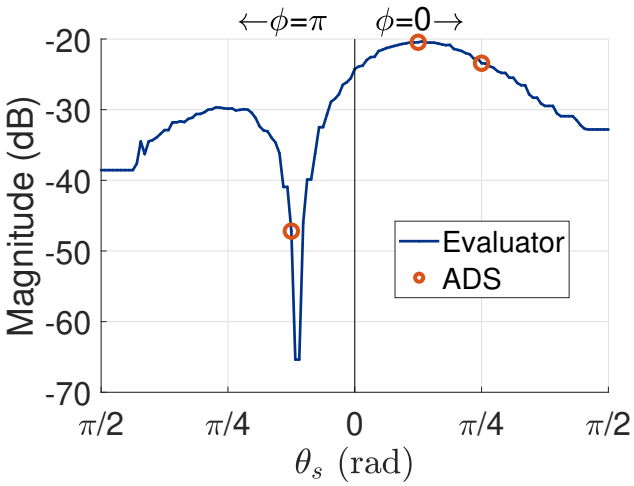
(a) Magnitude (b) Phase

Figure 2.17 Spatial responses for simulations performed using the evaluator simulation and Keysight ADS with 5-bit discrete phase shifters and unity gain amplifiers in an 8-element uniform linear array.

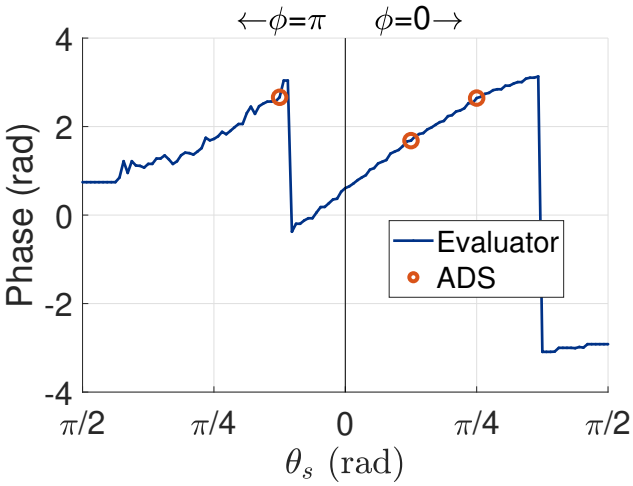


(a) Magnitude (b) Phase

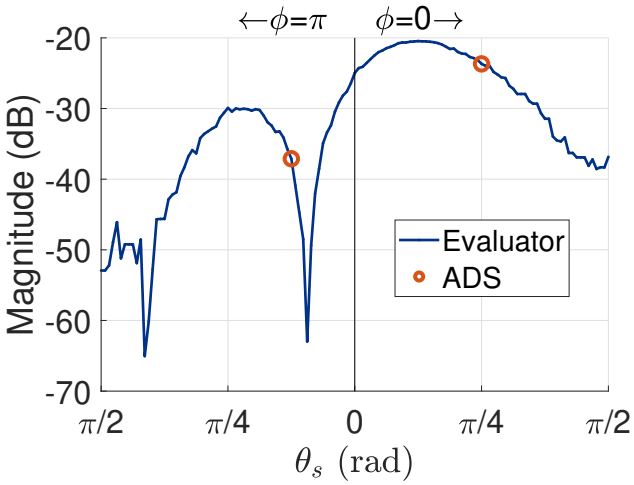
Figure 2.18 Spatial responses for simulations performed using the evaluator simulation and Keysight ADS with S-parameter based phase shifters and amplifiers for an 8-element uniform linear array.



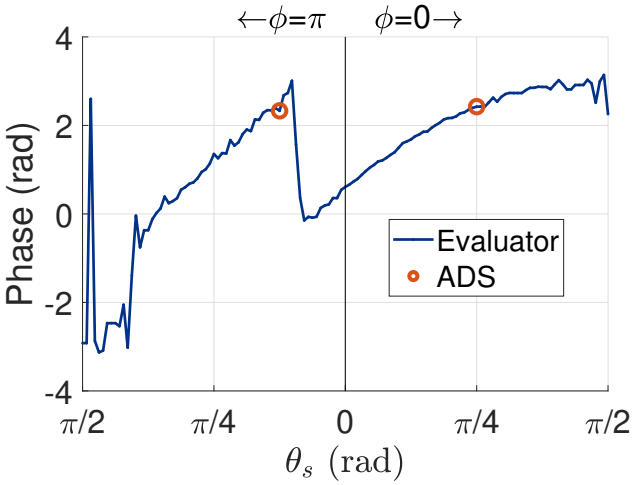
(a) Magnitude $\phi_s = 0$



(b) Phase $\phi_s = 0$



(c) Magnitude $\phi_s = \frac{\pi}{2}$

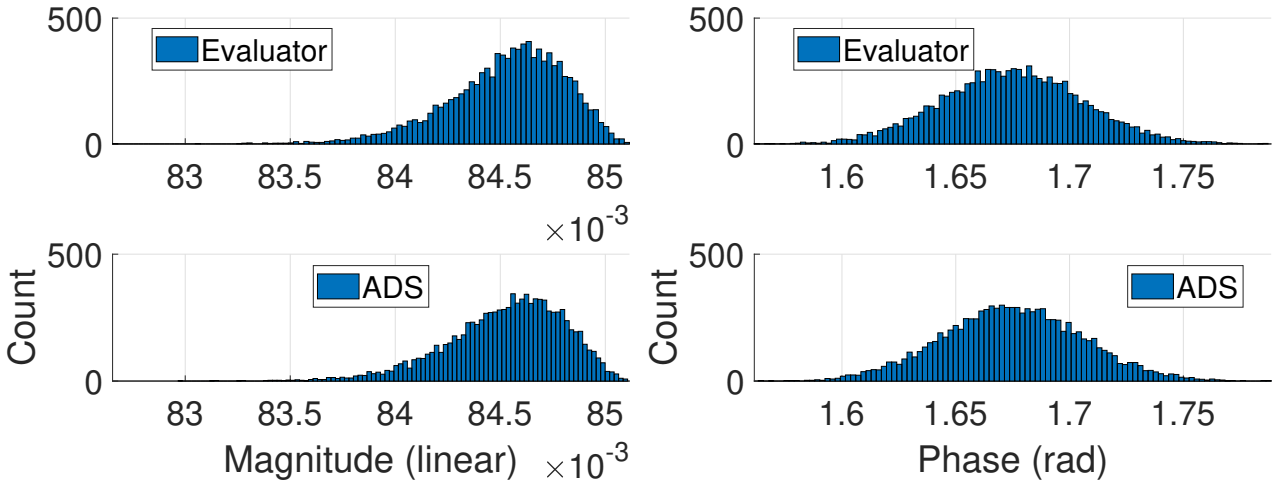


(d) Phase $\phi_s = \frac{\pi}{2}$

Figure 2.19 Spatial responses for simulations performed using the evaluator simulation and Keysight ADS with S-parameter based phase shifters and amplifiers for an 3x3 element planar array.

2.6.2 Verification against Commercial Software with Noise

Model noise was then introduced into both the ADS simulation and the developed evaluator. The verification tests with noise were run using the 8-element ULA phased array. Again, the receiver phased array beam was steered from $\theta_s = \frac{\pi}{2}$ to $\theta_s = 0.0$ with $\phi_s = \pi$ and $\theta_s = 0.0$ to $\theta_s = \frac{\pi}{2}$ with $\phi_s = 0.0$. The incident plane wave had angles of $\theta_i = \frac{\pi}{8}, \phi_i = \pi$. The model noise was added onto each phase shifter in the receiver phased array from a Gaussian distribution with a mean of 0.0 and a standard deviation of 5 degrees. Each of the simulations was run for a total of 10,000 Monte Carlo iterations. The complex responses for each of the Monte Carlo simulations with the steered array were then compared against the ADS simulation. The output distributions utilizing S-parameter-defined components can be seen in Figure 2.20(b) and Figure 2.20(a) for magnitude and phase, respectively.



(a) Magnitude (b) Phase

Figure 2.20 Distributions for array simulations using the evaluator and Keysight ADS with added phase noise and hardware models based on measured S-parameters. These distributions should be a Gaussian distributions about the nominal result due to the Gaussian phase error.

Due to the random sampling during the Monte Carlo simulation, the output of the Monte Carlo simulation cannot provide a quantitative comparison. The Kullback-Leibler (K.L.) divergence [87] was calculated from ADS and the evaluator's polar values to compare the results quantitatively. This K.L. value quantifies the relative differences between the evaluator distributions and the ADS distributions. The probability mass function (PMF) of each set of results was estimated by counting the number of results that fell into one of 500 discrete bins. The total was then normalized to one. The calculated divergences are given in Table 2.4. The K.L. divergence values are all less than 0.03 for magnitudes and 0.04 for phases. These small values show that these distributions are very similar, giving confidence that the evaluator is working correctly in

the presence of noise.

Table 2.4 Kullback-Leibler divergence of the evaluator from commercial software (ADS). Divergences were calculated separately for the magnitude and phase distributions.

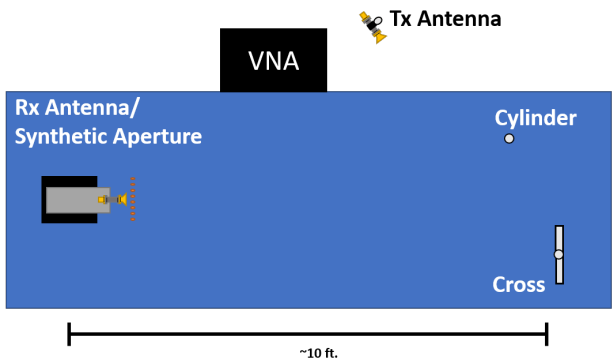
Configuration	**Magnitude (linear)**	**Phase (Radians)**
Continuous Phase Shifters	0.030	0.036
Discretized Phase Shifters	0.029	0.040
S-parameter Components	0.028	0.037

2.7 Verification against Measurements

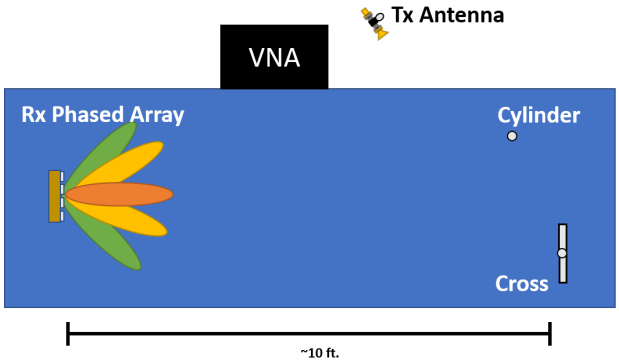
Alongside the other verification methods, attempts were also made to verify the evaluator against measurements. While the verifications attempts detailed in this section provide promising results, they did not provide concrete evidence of verification. This was not due to mistakes with the verification method or the evaluator, but with inconsistency in measurements and the inability to remeasure. During post-processing of the data, it was found that the measurements taken with the SAMURAI system were not sufficiently above the noise floor to provide accurate measurement of the second multipath component. This led to a mismatch in the final simulated result due to an innacurate measurement of the propagation channel.

The verification against measurements was performed by measuring a propagation channel with two scatterers in a bi-static configuration. First, the propagation channel was measured using the NIST SAMURAI system described in section 2.3.2 with a 16x16 element planar array with a spacing of 5.25 mm at about 28 GHz. The propagation channel was then measured using a millimeter-wave phased array from [88] by replacing the SAMURAI robot arm and steering the beams of the phased array. This array is an 8x8 array with a spacing of 5.25 mm operating at about 28 GHz. The layout of each of these measurements can be seen in Figure 2.21(a) and Figure 2.21(b). An image of the measurement configuration with the SAMURAI system can be seen in Figure 2.21(c) and the positions of each of the components in Figure 2.21(d).

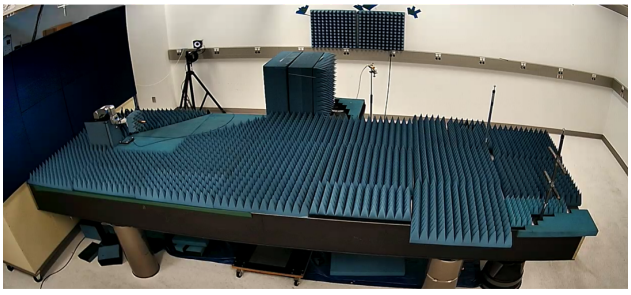
Using the SAMURAI measurements as the propagation channel response at each array element location in an 8x8 array, the hardware responses of the hardware phased array can be simulated and compared the output to that of the phased array. To provide uncertainty on the results due to error in position and measurement, multiple 8x8 subarrays were taken from different positions in the original 16x16 measurement. An example of this subarray can be seen in Figure 2.22. Using the evaluator, the response of the phase shifter and amplifier in the hardware phased array is simulated. The beam of this new simulated array is virtually steered in software to mimic the operation of the hardware phased array. By steering to the the same angles with the simulated phased array (and SAMURAI measurements) as were measured with the hardware phased array the output is compared to see the accuracy of the simulated hardware components. A flow chart of



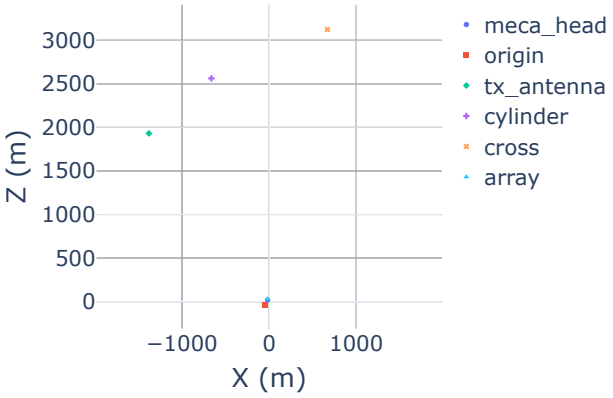
(a) Layout of the SAMURAI measurement.



(b) Layout of the phased array measurement.



(c) Image of the SAMURAI measurement.



(d) Channel configuration.

Figure 2.21 Setup of the measurement verification experiment.

this operation can be seen in Figure 2.23. This comparison was run with both measured (*sim_meas*) and parametric (*sim_par*, defined from the data sheet parameters) hardware models. In Figure 2.24(a), all of the spatial responses are shown for each stage of this experiment. Figure 2.24(b) shows a zoomed in version of this data without the SAMURAI measurements to provide better scaling.

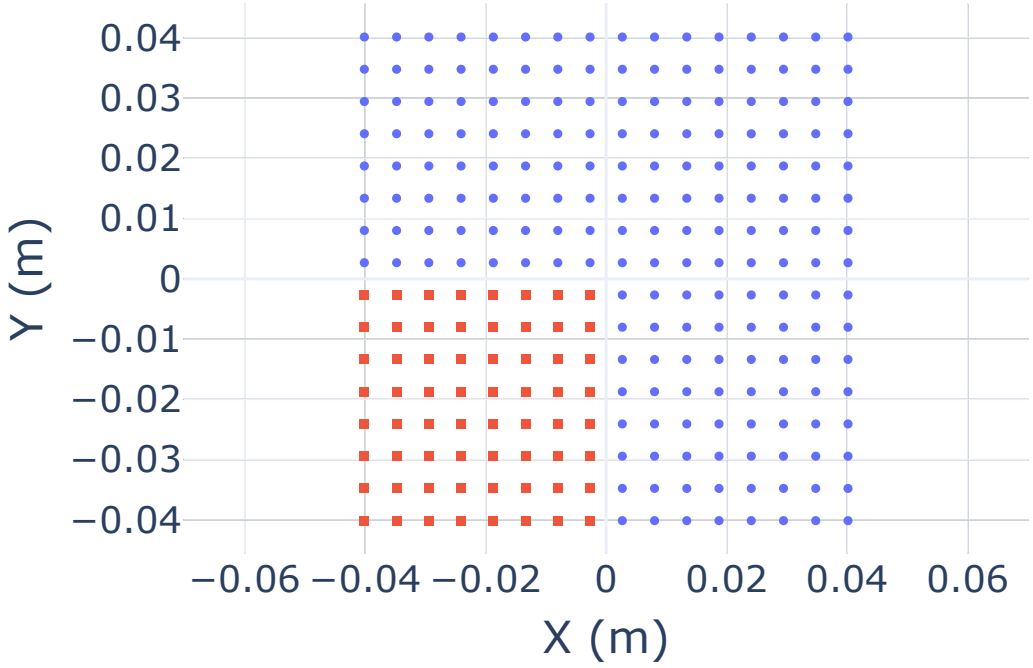


Figure 2.22 All positions measured by the SAMURAI system.

It is easily noticeable that the general shape, especially around the cylinder angle, is very similar between the measured and simulated phased array responses. The parametric simulated response has a similar shape, but is shifted down likely due to an inaccurate gain in the amplifier response. Near the cross angle, the response begin to drastically differ. This is likely due to the SAMURAI measurements themselves as the SAMURAI measurements have a null near this angle while the phased array measurements do not. It is theorized that the SAMURAI measurements may have been near the noise floor on this measurement whereas the hardware phased array provides amplification before the measurement system. Because the SAMURAI data itself seemed to be the cause of the unmatched results, this experiment was inconclusive and unable to be used to verify the evaluator. Further measurements were not able to be taken and therefore this experiment was not able to be repeated to test the theories on why it failed.

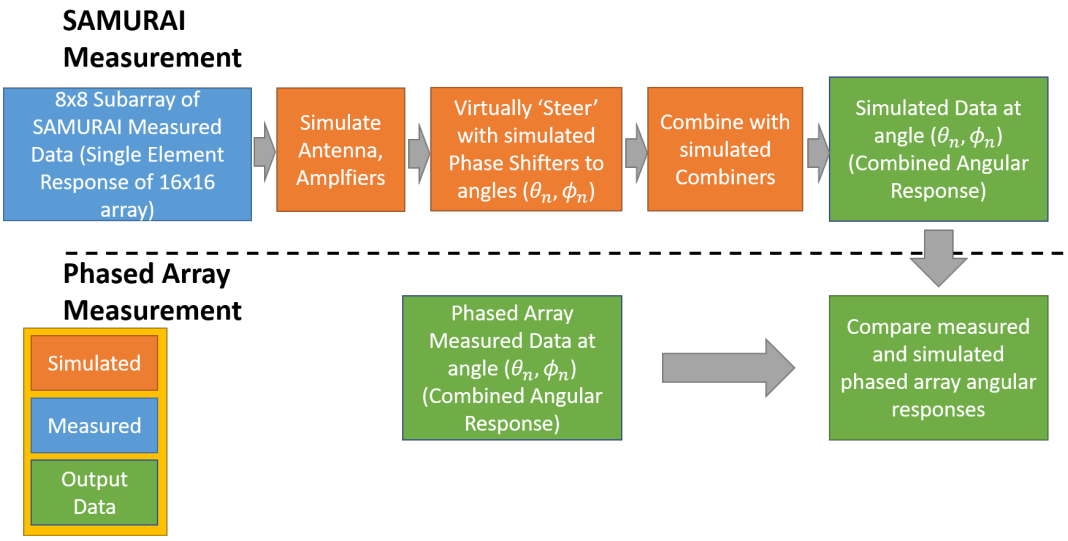
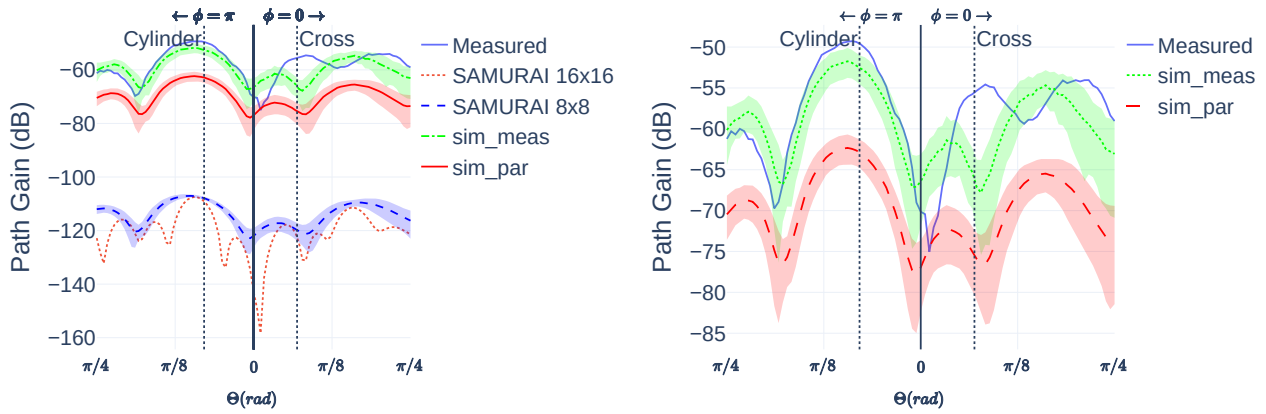


Figure 2.23 Flow chart of measurement verification.



(a) Spatial response of all measurements and simulated phased array outputs.

(b) Spatial response of simulated and measured phased array.

Figure 2.24 Results of measurement verification simulations and measurements.

2.8 Acceleration of the Channel Estimator Evaluator

The estimator evaluator was extended to provide distributed and accelerated simulation capability in the Python programming language. Because the evaluator is designed with scalability in mind, large arrays (such as 128×128 element planar arrays) can easily be defined and simulated. This allows for testing the limits of the evaluation complexity to find where acceleration techniques presented here can make infeasible problems feasible.

The problem focused on during acceleration consists of input data D_n with N_D complex data values that are packed into N_F different subcarrier frequencies across N_S different OFDM symbols. This means the last data value will be at subcarrier frequency f_{N_F} in symbol S_{N_S} so $D_{N_D} = D_{N_S N_F}$. These symbols are then transmitted from an isotropic antenna through a channel. The simulated channel is defined as a single path without reflection between a transmitter antenna and an $N_E x N_E$ element planar analog phased array. The array is assumed to be in the far field of the transmitter. The array therefore receives a single plane wave at normal incidence. Each element of the array consists of an isotropic antenna, a continuous phase shifter and a unity gain amplifier. A lossless combiner is simulated after these components to sum all the element values and provide simulated output symbols. The operations in this simulation consist mainly of matrix multiplication, trigonometric operations, exponentiation, and basic arithmetic operations with complex data types.

The runtimes of this simulation are very dependent on several variables from the number of elements in the array $N_E \times N_E$ to the number of subcarriers in each symbol N_S. When these values become large, the simulation becomes very computationally and memory intensive. Therefore, for larger simulations, it is important to utilize accelerated computing methods to reduce overall simulation times.

2.8.1 Vectorization

One method used to accelerate these simulations is through vectorization. This is done by utilizing libraries optimized to perform multiple simultaneous computations on a device. With a high-level programming language like Python, vectorization is typically achieved by grouping function inputs in memory and making one function call to process a large group of data. To get the most out of this optimization, multiple symbols are simulated together as a symbol group.

Vectorization on a central processing unit (CPU) is performed by ensuring all possible operations were called using the Intel Math Kernel Library (MKL) [64] which provides hardware optimized function calls. In Python, computationally intense sections are calculated on grouped inputs using Numpy [89] or Numba [90] calls which in turn call MKL. More in depth information and runtimes on vectorization with these libraries is given in [62].

Graphics processing units (GPUs) extend on this vectorization by providing hundreds or thousands of simultaneous computation units. GPU Vectorization was performed with Nvidia's compute unified device architecture (CUDA) [91] and the CuPy [92] library. Like with the CPU, this is achieved by grouping together function inputs and performing computationally heavy simulation sections with CuPy functions. Even further acceleration can be achieved through optimization with custom CUDA kernels.

2.8.2 Distributed Parallelization

While vectorization alone reduces simulation runtimes, it is limited by the memory and computational units available to a single device. By parallelizing across multiple CPUs (or CPU cores) or GPUs, simulation memory can be distributed to calculate larger arrays and more symbols simultaneously. Computation times can also be increased by combining the total computational power of multiple discrete devices. These calculations are performed across M different devices.

By parallelizing simulations across dimensions that are independent, no communication is required during runtime between discrete devices. The OFDM simulation used assumes no inter-symbol-interference (ISI) between OFDM symbols and no crosstalk between hardware components. This presents multiple methods to parallelize the simulation across devices. One is by splitting across each array element in the simulation. The other is by splitting across the grouped symbols that are calculated simultaneously.

2.8.3 Parallelization Across Array Elements

Parallelizing across array elements allows simulation of arrays that are too large to fit on memory limited devices such as GPUs. It also accelerates computations by simultaneously computing separate sections of the array on different devices. With this method, the simulated elements in the phased array system are split across devices such that each device is simulating only a subsection of the system. Input symbol groups are then simulated for each of these subarrays. The output of each of these separate simulations is then gathered and combined to produce the correct output. A representation of this can be seen in Figure 2.25.

2.8.4 Parallelization Across Symbols

Parallelizing across symbol groups allows for simultaneous simulation of large symbol groups. Unlike parallelization across array elements, each device in this scenario simulates the entire phased array system. The symbol group to be simulated is split across the devices and the simulation is performed. The outputs from each device are then stacked to provide the complete output simulation of all input symbols. This method of parallelization is geared best for simulations of smaller arrays (e.g. 16x16) where large numbers of simulated symbols are desired. This does come at the cost of being memory limited for extremely large arrays. A representation of this method can be seen in Figure 2.26.

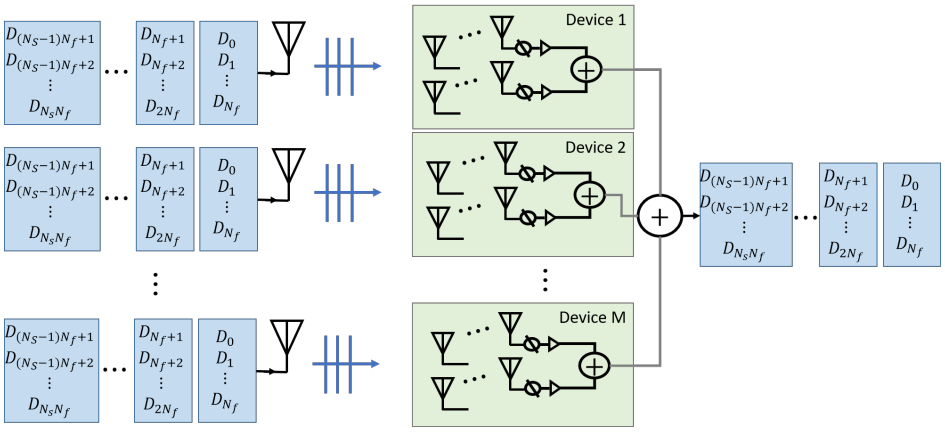


Figure 2.25 Graphical representation of parallelization across array elements.

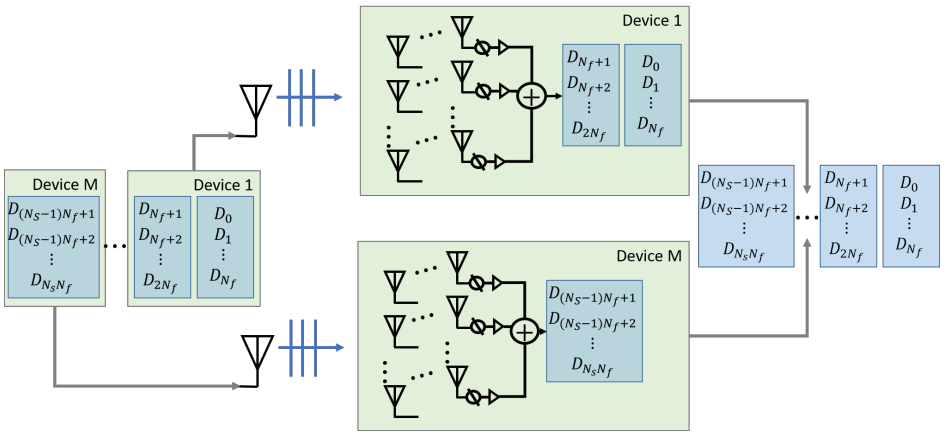


Figure 2.26 Graphical representation of parallelization across symbol groups.

2.9 Results of Acceleration

Single CPU and GPU implementations using vectorization and multi-GPU and CPU implementations using distributed parallelization were timed for varying simulation configurations. Runtimes were gathered when parallelizing across array elements and input symbol groups. Each of these results was performed on an Intel i7-4790 CPU with up to 8 cores with 32 GB of RAM and up to 2 Nvidia RTX 2080 GPUs with 8GB of memory each. These results are shown on both linear and logarithmic scale for visual clarity.

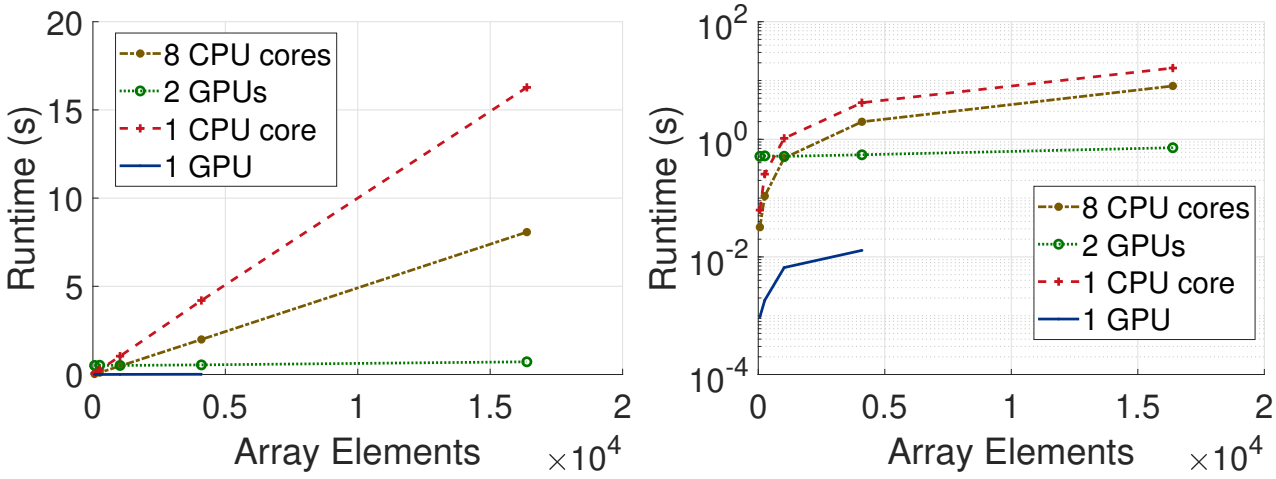
2.9.1 Parallelization Across Array Elements

The first set of tests kept a constant amount of input symbols, simulating 500 OFDM symbols each with 200 subcarriers per symbol. The size of the array was then swept to simulate rectangular planar arrays with 8x8, 16x16, 32x32, 64x64, and 128x128 elements. Parallelization for these tests was performed across the array elements due to the relatively small symbol group size and large possible array sizes.

Figure 2.27 shows the results of these simulations across each of the array sizes. Focusing on CPU performance, it can be seen that by parallelizing the simulation across 8 different CPU cores, the runtime is reduced by more than half. Each of these curves has a linear trend that would likely continue if the number of elements were increased on a system with enough memory. As expected, the runtimes of the GPU implementation are much lower than the CPU. Unexpectedly, the 2 GPU case was slower than the single GPU case. Further investigation found that this was an unfortunate side effect of the Python GPU interface. To properly run the multiple GPUs in a non-blocking way, each GPU had to be controlled separately individual processes using the python *multiprocessing* library. Because the operations on a single GPU were so highly vectorized and were completed so quickly, as multiple GPUs are used there is actually a slowdown due to a communications bottleneck. This is because data must first pass to the processes and then to each of the GPUs as opposed to directly being passed to the GPU from a single python process. Further work could reduce this bottleneck by implementing a custom C or FORTRAN interface to directly interact with the GPUs. This would remove most of the overhead seen by using *multiprocessing*, but would increase the complexity of the code. In this scenario, a CUDA kernel would need to be written for each hardware response whereas the python interface allows *numpy* operations to be directly swapped for *cupy* operations with no additional code. Even with this communication bottleneck, all GPU implementations drastically outperform all CPU implementations. One limitation of the single GPU not seen in multiple GPUs is the amount of memory it has. This drastically limits the size of the array that can be simulated. This can be seen by the fact that the GPU was unable to compute past an array with just under 500 elements with the given number of subcarriers and symbol size. By moving to two GPUs this memory limitation can be somewhat overcome. This allows for utilizing the memory across multiple GPUs for larger simulations.

Because the GPU simulation is limited by communication, there is a decrease in throughput because the host must serially send data to 2 devices. Larger arrays could be simulated on memory limited device like GPUs by reducing the symbol group size or number of subcarriers, at the cost of reducing some vectorization gains if large numbers of symbols are needed to be simulated.

It is expected that increasing the number of CPU cores beyond what was tested here would continue to decrease the runtime of the simulation. As the number of CPU cores approaches the number of elements in the array, the simulation will likely start to become memory limited, reducing the gains seen from an increased parallelization. As the number of GPUs utilized is increased, it is likely there will continue to be slightly reduced performance due to the communication overhead with a larger number of discrete devices. Depending on the simulation, these increases in runtime may be offset by the larger available for computations.



(a) Linear scale

(b) Logarithmic scale

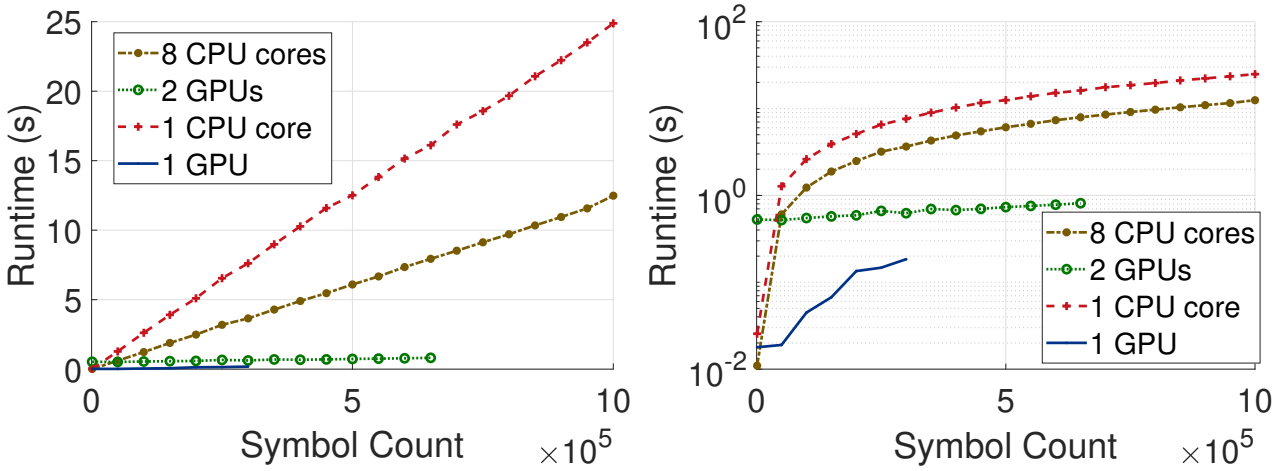
Figure 2.27 Runtime vs. array size using vectorization and parallelization across array elements.

2.9.2 Parallelization Across Symbols

Another test was run with varying sizes of input symbol groups and a fixed planar array with 16x16 elements (256 total elements). The size of the symbol group varied from 5 to 5000 symbols each with 200 subcarriers. For this case parallelization across symbols is performed.

Figure 2.28 shows the runtimes of single CPU/GPU and multi-GPU/CPU implementations. Again, parallelizing across 8 CPU cores outperforms the single core implementation which only uses vectorization. The GPU implementations again drastically outperform any CPU implementations but are unable to perform the same size of simulations due to memory limitations. By using multiple GPUs, some of the memory

limitation can be overcome with minimal loss in acceleration due to communications overhead. For large symbol scenarios, the GPU implementations could again perform multiple block calculations at the expense of vectorization efficiency. Again, it is expected that increasing the number of CPU cores would continue to decrease runtime, but will become memory limited if the simulation size is not increased. Adding more GPUs, greater runtimes would be expected due to increased communications, but with the added benefit of additional memory for larger simulations.



(a) Linear scale

(b) Logarithmic scale

Figure 2.28 Runtime vs. array size using vectorization and parallelization across array elements.

2.10 Convergence of Evaluator

Before using the evaluator, a final step was taken to ensure accurate evaluation. Because the evaluator takes random samples from the noise distributions of hardware components during each Monte Carlo iteration, it must be ensured that the result from the evaluator has converged to an acceptable level. This convergence level determines how much error there is in the evaluation. Therefore, convergence testing was performed for a number of scenarios to provide a baseline of the approximate error for different estimators and hardware components. Convergence testing also has the side effect of providing information on how repeatable different estimators are for each hardware configuration.

These evaluator convergence tests were run with the following configuration:

- Channel - Single plane wave incident at boresight
- Bitstream - 2^{11} random bits
- Modulation - 16QAM

- Pilots - pilot tone every 10 subcarriers
- Frequency - 100 subcarriers spaced every 15 kHz starting at 27.5 GHz
- Phased Array - Continuous phase shifters, unity gain amplifiers, lossless combiners

.

Evaluation using this configuration was then repeated $R = 50,000$ times ($r = [1, ..., 50000]$), each producing a metric M for MSE, EVM, and BER each of length R. To calculate the convergence of the error, subsets of these R iterations of different sizes S ($s = [1, ..., S]$) with O offsets τ_o ($o = [1, ..., O]$) were taken. The mean of each of these different subsets (for each different size) were then taken and the standard deviation of these means was taken. This calculation gives the convergence of the mean value of a subset of a given size. This operation can be written mathematically for each offset o and size s as:

$$\mu_{os} = \frac{\sum_{s=1}^{S} M[s + \tau_o]}{S}. \tag{2.23}$$

Taking the standard deviation for each offset then provides the standard deviation for each subset size σ_s

$$\sigma_s = \sqrt{\frac{1}{O}\sum_{o=0}^{O}(\mu_{os} - mean(\mu_{os}))}. \tag{2.24}$$

This standard deviation in the subset means σ_s helps provide an idea of how much error may occur in the evaluation given number of repeats R. Careful selection of R ensures the error is not too high to accurately compare estimators. This test with subsets of up to $S = 5000$ for MSE and EVM can be seen in Figure 2.29(a) and Figure 2.29(b), respectively. The figure for BER is not provided as the magnitude of the error was low with respect to the modulation scheme and therefore no bit errors were recorded ($BER = 0$ for these cases).

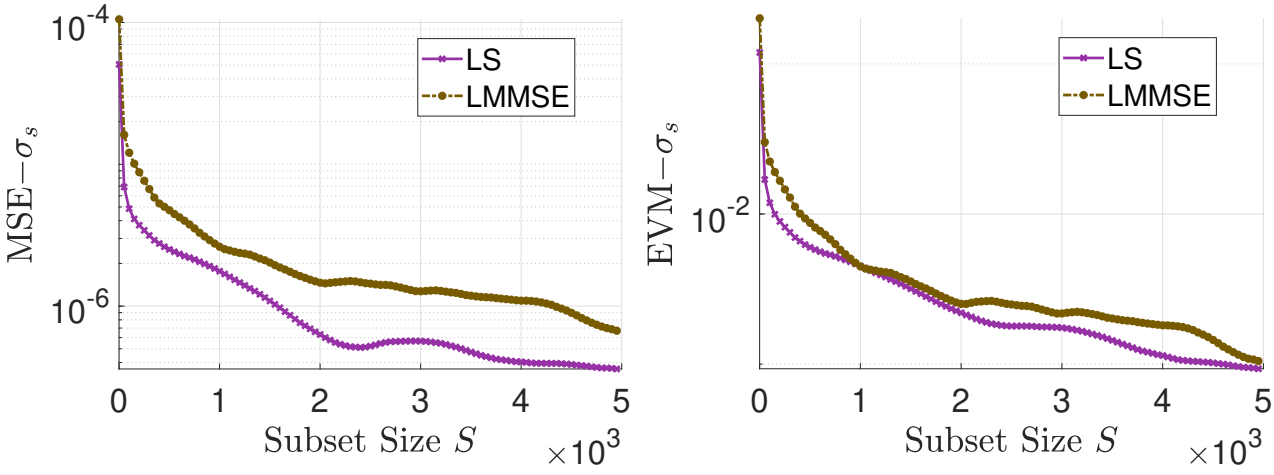
When evaluating channel estimators, a number of repeats equal to the subset size S are repeated, corresponding to the amount of error deemed allowable. For many cases this is a fine balance between the time it takes to evaluate an estimator and the accuracy of the evaluation.

2.11 Evaluation of a Least Square Channel Estimator

The evaluator was first tested using a basic least square channel estimator to test basic functionality. The LS estimator [21] can be written as

$$h_f = \frac{y_f}{x_f} \tag{2.25}$$

where y_f is the output of the pilot tones, h_f is the channel response, and x_f is each of the input pilot values (see 1.4.1 for more information). The frequency response is estimated at each subcarrier frequency that has a pilot tone. There exist many possible spacings of pilots such as those discussed in [93, 94]. This work focuses



(a) MSE Convergence

(b) EVM Convergence

Figure 2.29 Convergence of metric evaluation with different numbers of repeats (subset sizes S) as given by (2.24).

on a comb-type spacing [24] with a pilot tone every 25 subcarriers. After the response of each subcarrier with a pilot is estimated, a 2D linear interpolation is performed using the Scipy griddata function to estimate the response at any subcarrier frequencies with no pilot tones. Any subcarriers that require extrapolation are estimated using the nearest neighbor approach. The partial implementation of this estimator can be seen in Listing B.9.

The least-square estimator was evaluated with two different receiver's phased array configurations, *Array 1* and *Array 2*. Both phased array configurations consisted of an 8x8 element planar array uniformly spaced at 5.35mm. The phased arrays used unity gain amplifiers, continuous phase shifters, and lossless combiners. The noise on the amplifiers on both arrays were set to have a linear standard deviation of 0.01 (-20dB) with a mean of 0.0. The phase shifters on *Array 1* have a 5-degree standard deviation and a mean of 0.0. The phase shifters on *Array 2* have a 15-degree standard deviation and a mean of 0.0. A lossless combiner was used in both phased arrays. Other configurations not shown here with varying noise levels were also tested with similar outcomes.

Evaluation was performed using *Array 1* and *Array 2* using a 16- (see 2.2) and 256-QAM as specified by 3GPP [95]. A bitstream of 2^{18} bits run over 100 subcarriers was used. The calculations of the system-level metrics were the mean of 1000 independent simulations. Configurations with different arrays and modulations were run to demonstrate the importance of the metrics chosen and how each is affected by different parts of communications systems. Table 2.5 contains the results of these evaluations.

MSE is affected by the increased hardware noise, but not by the modulation. As in Eq. (5), MSE is calculated from the channel estimate H_{est} which is dependent only on the communication system response

Table 2.5 Metrics from the evaluator when evaluating a least square channel estimator. Results of the evaluated metrics are provided as *Nominal(St.Dev.)*.

	Array 1		**Array 2**	
	16-QAM	**256-QAM**	**16-QAM**	**256-QAM**
MSE	6.54×10^{-4} (6.21×10^{-6})	6.43×10^{-4} (8.71×10^{-6})	3.01×10^{-3} (3.22×10^{-5})	3.01×10^{-3} (4.42×10^{-5})
EVM	2.61% (0.013%)	2.61% (0.017%)	0.029% (5.60%)	5.60% (0.042%)
BER	0 (0)	1.43×10^{-4} (2.53×10^{-5})	0 (0)	1.47×10^{-2} (3.92×10^{-3})

and the pilot tone values. EVM, as defined in Eq. (6), is affected by hardware noise but not modulation assuming the normalization factor $\sum_{n=0}^{N} |D|^2$ stays constant. BER is affected by both the modulation and hardware noise because it is dependent on the spacing of constellation points for a given modulation. BER fails to capture the differences between *Array 1* and *Array 2* for 16-QAM due to the binary nature of the metric.

Required average EVM levels for each modulation scheme are outlined in the 3GPP specification for 5G NR [96, p.171]. A list of these EVM levels are given in Table 2.6. Based upon these requirements, *Array 1* meets the requirements to run at both 16-QAM ($EVM < 12.5\%$) and 256-QAM ($EVM < 3.5\%$). *Array 2*, however, only meets the requirements for 16-QAM but not for 256-QAM due to the increased noise from the phase shifters. Other types of channel estimators can account for some of this noise [21, 93] and may reduce the EVM, allowing for noisier phased array hardware without reducing the quality of service. The estimator evaluator allows this reduction in EVM and other metrics to be quantified.

Table 2.6 Required maximum EVM for each modulation scheme as specified in 3GPP 138.101.1.

Modulation Scheme	**EVM**
Pi/2-BPSK	30%
QPSK	17.5%
16-QAM	12.5%
64-QAM	8%
256-QAM	3.5%

CHAPTER 3
MONTE CARLO AUGMENTED CHANNEL ESTIMATOR

A novel Monte Carlo augmented channel estimator (MCCE) was developed which can provide a lower mean error and standard deviation in its estimation than other channel estimators currently in use. This estimator uses *a priori* knowledge of hardware components within a communications system to improve the estimate of a systems noise distribution on input data when compared to other methods. This improved noise estimation then in turn provides the reduced mean error and the reduced standard deviation to reduce the MSE, EVM, and BER which can increase system throughput.

As discussed previously in section 1.4, there exists a large amount of published works on channel estimation as a whole and estimation for 5G mmWave systems. Previous authors have also developed methods and techniques to improve channel estimation in communications systems using Monte Carlo methods [97, 98]. However, these efforts do not focus on the hardware of the large arrays seen in 5G mmWave communications systems.

With the MCCE, these previously developed Monte Carlo channel estimation schemes are improved upon. The MCCE is designed to reduce the error and deviation in channel estimation metrics in 5G mmWave systems with electrically large phased arrays using *a priori* knowledge of the frequency responses and error characteristics of phased array hardware components such as power amplifiers and phased-shifters. By reducing the bit errors from the channel, the MCCE can provide increased throughput in a communications system.

3.1 Monte Carlo Augmented Channel Estimator Design

The MCCE improves the accuracy of current channel estimators by supplementing received signals with simulations of phased array hardware components to help account for deviations from a nominal solutions (i.e. referred to as *statistical bias*) in the phased array. Noise distributions of the combined hardware are generated by randomly sampling each distribution (e.g. phase and magnitude noise) of individual hardware components. The estimation of a combined distribution is not typically attainable through analytical calculation due to the complexity and therefore Monte Carlo methods are relied upon. In the following section, estimated variables are denoted with hats (e.g. $\widehat{x}$) and vectors are denoted as bolded variables (e.g., $\mathbf{x}$).

The Monte Carlo augmented channel estimator begins like any other OFDM channel estimation scheme. A set of pilot tones (predetermined values) are first transmitted at the frequencies of interest. A receiver then picks up these pilot tones along with the effects of both the channel and the system hardware. For basic problems this can be represented by

$$y_f = (h_f + n_f)x_f \tag{3.1}$$

for a frequency domain estimation where y_f is the received value, h_f is the channel response with noise n_f, and x_f is the known set of pilot tones for a subcarrier f. By performing channel estimation, the response is estimated from the input to the output signal given by h_f. This estimate of the response can then be used to correct subsequently transmitted data.

Unfortunately, when looking more in depth at the system, the problem becomes much more complicated. The channel response h_f can be represented by a number of different cascaded responses from each part of the system. This can be represented in an expanded form by N different cascaded hardware components and propagation channels where each component or channel has its own systematic and random (noise) uncertainties. This can be represented in the frequency domain as

$$y_f = (h_{f0} + n_{f0})(h_{f1} + n_{f1}) \cdots (h_{fM} + n_{fM})x_f, \tag{3.2}$$

where $(h_{mn} + n_m)$ is the response and error model of the m^{th} component between the transmitted and received signal where $m = [0, \cdots, M]$. In a real system, the uncertainty mechanisms can widely vary for each part of the system and be anything from a uniform distribution to nonlinear effects from sources like round-off errors from discretized phase shifters. This expanded form of channel estimation depicts how multiple linear and non-linear effects can compound to provide a bias in the total estimated channel h_f. Using Monte Carlo analysis, this bias can be estimated and used to provide a better correction for the transmitted data. This estimation requires pre-determined models for each part of the system that will be used in the estimation. These models will vary depending on the communications system being estimated, but each will have a nominal response which will be denoted by h_{fm} and some corresponding set error mechanisms n_{fm} as described in section 2.2.2. With the initial channel estimation performed, the only unknown response in the chain is that of the wireless channel. Using the nominal responses from each of the defined models, the expected nominal response of any unmodeled parts of the system (e.g. a propagation channel) can be approximated as $\hat{\mathbf{h}}_{umod}$

The statistical bias that may occur from uncertainties in hardware models can be demonstrated through the simple example of the effect of angular accuracy of a phased array. For this example it is assumed there is a linear and equally spaced phased array with a known incoming incident wave at azimuth angle θ_0. The assumption is also made that there is an error mechanism in the communications system that adds a Gaussian distributed angular error $\pm\theta_{err}$. Because the angular error is Gaussian, its nominal is simply given as its mean. If a large number of measurements, N, are taken of θ_n where $n = 1, 2, \cdots$, it is expecteded the

mean of those values to give the correct angular value as N continues to grow. Unfortunately, beamforming transformation for received magnitude is non-linear. The magnitude of the nominal received value v_0 would be when the array points directly to θ. Again, if N measurements are taken with received values v_n for $n = 1, 2, \cdots$, as N increases it is expected that the mean of all of the measured angles θ_n to equal θ_0. The magnitudes on the other hand are at a maximum when $\theta = \theta_0$ and therefore $v_{1,2,\ldots} \leq v_0$. Because of this it is not expected that the mean to be equal to the nominal value and therefore the estimate will have some bias to it.

The MCCE assumes the transmitter hardware, propagation channel responses, and receiver antennas responses are unknown. These unknown (or unmodeled) responses at each pilot tone subcarrier are given as the vector $\widehat{\mathbf{h}}_{\text{umod}}$. In contrast, the MCCE's receiver phased array response, $\mathbf{h}_{\text{mod}}$ is fully modeled by a phase shifter and amplifier behind each antenna element n, with a combiner at the output of each phased array element for each frequency. Again, each of these hardware component models that make up $\mathbf{h}_{\text{mod}}$ consist of a nominal frequency response and an error distribution. Each Monte Carlo iteration samples from these distributions to provide the total hardware response for that iteration. These responses and distributions of these models are defined by either a mathematical formulation or measured S-parameter values. These models are the same as those used in the evaluator in chapter 2. Using $\mathbf{h}_{\text{mod}}$ and $\widehat{\mathbf{h}}_{\text{umod}}$ together, a distribution of possible received values $\widehat{\mathbf{y}}_{\text{mc}}$ is then generated and used to estimate the channel. A flow chart of the MCCE operation is shown in Figure 3.1 and the base definition for this channel estimator can be found in Listing B.10.

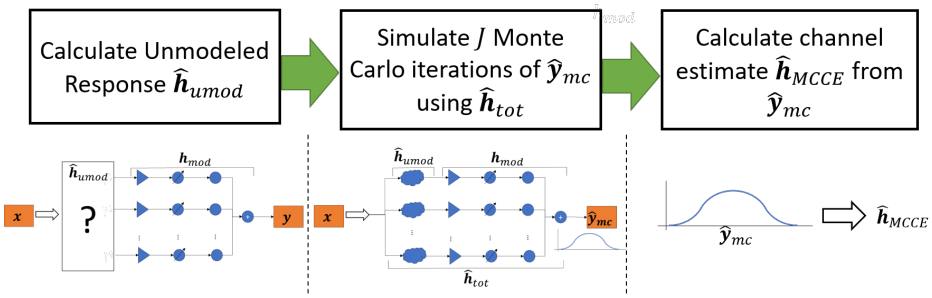


Figure 3.1 Flow chart of the MCCE operation.

3.1.1 The Modeled Response

The MCCE relies on known models of the phased array hardware, as in section 2.2.2. The combined response of each of these models is the modeled response $\mathbf{h}_{\text{mod}}$. The modeled response is used by the MCCE for its channel estimation. The estimator is, therefore, unique to the configuration of each communications system and phased array. The models are provided as *a priori* knowledge into the estimator for each unique

phased array hardware configuration. Once the hardware models have been defined, the MCCE uses a multi-step operation to provide a channel estimate given a set of input signals $\mathbf{x}$, and output signals $\mathbf{y}$, at each subcarrier f.

3.1.2 Estimating the Unmodeled Response

For the configuration used here, it is necessary to estimate the unmodeled response at each element in the received phased array. Repeat samples of the MCCE phased array hardware models (i.e. the modeled response $\mathbf{h}_{mod}$) generate K ($k = [1, \ldots, K]$) total possible responses at each antenna element. The unmodeled response $\widehat{\mathbf{h}}_{\text{umod}}$ can therefore be written for each phased array element n at each subcarrier f as

$$\begin{bmatrix} x_f h_{\text{mod}\,f11} & \cdots & x_f h_{\text{mod}\,fN1} \\ \vdots & \ddots & \vdots \\ x_f h_{\text{mod}\,f1K} & \cdots & x_f h_{\text{mod}\,fNK} \end{bmatrix} \begin{bmatrix} \widehat{h}_{\text{umod}_{f1}} \\ \widehat{h}_{\text{umod}_{f2}} \\ \vdots \\ \widehat{h}_{\text{umod}_{fN}} \end{bmatrix} = \begin{bmatrix} y_{f1} \\ y_{f2} \\ \vdots \\ y_{fK} \end{bmatrix}, \tag{3.3}$$

where $y_{f1} = y_{f2} = \ldots = y_{fK}$ is the received pilot value (or mean of multiple pilot values) and $\widehat{h}_{\text{mod}_{fnk}}$ is the modeled response of the amplifier, phase shifter, and combiner at element n. The value of $\widehat{h}_{\text{umod}_{fn}}$ at each subcarrier is then calculated using a least-squares approximation of the over-determined system of equations. This results in an estimate of the unmodeled response of the communications system at each receiver phased array element n. The implementation of solving for the un-modeled response used in this dissertation can be seen in Listing B.11.

3.1.3 Estimating the Total Response

The unmodeled response $\widehat{h}_{\text{umod}_{fn}}$ is then combined with the modeled responses at each frequency h_{mod_f} to get a total response $\widehat{h}_{\text{tot}_f}$. J output values $\widehat{\mathbf{y}}_{mc}$ are simulated using a given input x_f. The LS channel estimator is used to estimate the total response for each iteration $\widehat{h}_{\text{tot}_{fj}}$, where j is the Monte Carlo iteration $j = [1, \ldots, J]$. The relationship of each of these values can be described in a matrix format as

$$\begin{bmatrix} \widehat{h}_{\text{umod}_1} h_{\text{mod}_{11}} & \cdots & \widehat{h}_{\text{umod}_1} h_{\text{mod}_{1J}} \\ \vdots & \ddots & \vdots \\ \widehat{h}_{\text{umod}_F} h_{\text{mod}_{F1}} & \cdots & \widehat{h}_{\text{umod}_F} h_{\text{mod}_{FJ}} \end{bmatrix} = \begin{bmatrix} \frac{\widehat{y}_{mc_{11}}}{x_1} & \cdots & \frac{\widehat{y}_{mc_{1J}}}{x_1} \\ \vdots & \ddots & \vdots \\ \frac{\widehat{y}_{\text{mc}_{F1}}}{x_F} & \cdots & \frac{\widehat{y}_{\text{mc}_{FJ}}}{x_F} \end{bmatrix} = \begin{bmatrix} \widehat{h}_{\text{tot}_{11}} & \cdots & \widehat{h}_{\text{tot}_{1J}} \\ \vdots & \ddots & \vdots \\ \widehat{h}_{\text{tot}_{F1}} & \cdots & \widehat{h}_{\text{tot}_{FJ}} \end{bmatrix} \tag{3.4}$$

where $\widehat{h}_{tot_{fj}}$ is the relation between x_f and $\widehat{y}_{\text{mc}_{fj}}$ for subcarrier $f = [1, \ldots, F]$.

3.1.4 Estimating the Channel

After the values of $\widehat{\mathbf{h}}_{\text{tot}_j}$ are generated, multiple approaches are possible to get the final channel estimate $\widehat{\mathbf{h}}_{\text{MCCE}}$ at subcarrier f. The first approach is to take the mean across all values of j yielding

$$\widehat{\mathbf{h}}_{\mathrm{MCCE}_f} = E[\widehat{\mathbf{h}}_{\mathrm{tot}_f}] = \frac{1}{J}\sum_{j=1}^{J}\frac{\widehat{y}_{\mathrm{mc}_{fj}}}{x_f}. \tag{3.5}$$

In essence, this is a modified LS estimator that has an increased number of samples available from the Monte Carlo simulations. This approach is simple and therefore does not add even more computational complexity to the MCCE. This approach is utilized for the results shown in this dissertation. The code for both calculating the total response and estimating the channel using the MCCE is found in Listing B.12.

3.1.5 Alternative Approaches

Another possible approach for the channel estimation uses all estimated values of $\widehat{\mathbf{h}}_{tot_j}$ to approximate the auto-correlation matrix and noise variance of the response. These estimated values could then be used as inputs for a LMMSE estimator. This approach removes the need to know the auto-correlation matrix and noise variance directly from other means (e.g., measurements) prior to channel estimation. It is worth noting that while both the LMMSE and the MCCE require *a priori* knowledge, the MCCE only requires system hardware models. Any hardware that is not modeled will be estimated in the unmodeled response. The LMMSE estimator requires knowledge of the phased array hardware and any values that may change over time (e.g., the propagation channel). This means that exhaustive knowledge must be known for all permutations of hardware states (e.g. for steering a phased array beam) and propagation channel states (e.g. a new machine added to a factory floor). The augmentation of the LMMSE with Monte Carlo methods creates a more practical estimator for a realistic system, at the cost of computational complexity.

3.2 Evaluation Configuration

In the following sections, channel estimators were evaluated with QPSK, 16-, 64-, 256-, 1024-, and 4096-QAM modulation schemes. Each estimator was tested with 100 subcarriers in each OFDM symbol. Each symbol contained pilot tones that were evenly spaced at intervals of 10 subcarriers. The responses at non-pilot subcarriers were calculated from the estimated response of each pilot tone using a two-dimensional (2D) linear interpolation and nearest neighbor extrapolation across frequency and symbols (time). The channel estimators were evaluated in multiple propagation channels (with polar angle $\theta = (0, \pi)$ and azimuthal angle $\phi = (0, 2\pi)$ as described in appendix A) as follows:

Propagation Channel 1 *(Boresight)* – One incident plane wave at $\theta_0 = 0, \phi_0 = 0$ and a magnitude of 0 dB.

Propagation Channel 2 *(Double Multipath)* – Two incident plane waves at $\theta_0 = \frac{\pi}{4}$, $\phi_0 = 0$ and a magnitude of 0 dB and $\theta_1 = \frac{\pi}{4}$, $\phi_1 = \frac{5\pi}{4}$ with a magnitude of -10 dB.

Propagation Channel 3 *(SAMURAI/Measured)* – A measured SAMURAI (section 2.3.2) channel in a bi-static radar configuration with reflections from two 1 inch diameter metal rods.

These channels are coupled with simulated 8x8 planar receive phased arrays to test each channel estimator. Each phased array consisted of an isotropic antenna and ideal (lossless and equal) combiner with differing amplifiers, phase shifters. Phase and amplitude noise were added on top of the phase shifters and amplifiers to capture the effects of noise on the estimator. In some cases, parameters (e.g. noise) of these arrays were swept (e.g. to vary SNR). In others (typically for preliminary testing), static values were used for these arrays. These arrays with static values were defined as follows:

Phased Array 1 *(Generic MCCE)* – Continuous phase shifters (standard deviation 10 degrees), unity gain amplifiers (standard deviation 2dB).

Phased Array 2 *(Generic MCCE Medium Noise)* – Continuous phase shifters (standard deviation 20 degrees), unity gain amplifiers (standard deviation 2dB).

Phased Array 3 *(Generic MCCE High Noise)* – Continuous phase shifters (standard deviation 50 degrees), unity gain amplifiers (standard deviation 2dB).

Phased Array 4 *(Discrete 5bit MCCE)* – 5-bit discretized phase shifters (standard deviation 10 degrees), unity gain amplifiers (standard deviation 2dB)

Phased Array 5 *(S-parameter)* – S-parameter defined phase shifters with 32 possible states (5-bit) (standard deviation 5 degrees), S-parameter defined amplifier (standard deviation 3.5dB).

The MCCE was evaluated with *Phased Array 1*, *Phased Array 2*, and *Phased Array 4* as the modeled response. Each estimator was evaluated in communications systems with *Phased Arrays 1-5*. Multiple OFDM symbols were provided to each estimator during testing. It is assumed that the total system response does not vary from symbol to symbol, and therefore the noise is the only difference in the values of subcarriers between symbols. LS and LMMSE calculate the mean at each subcarrier across the symbols. The MCCE uses the mean value of each subcarrier to calculate the unmodeled response $\widehat{\mathbf{h}}_{\text{umod}}$.

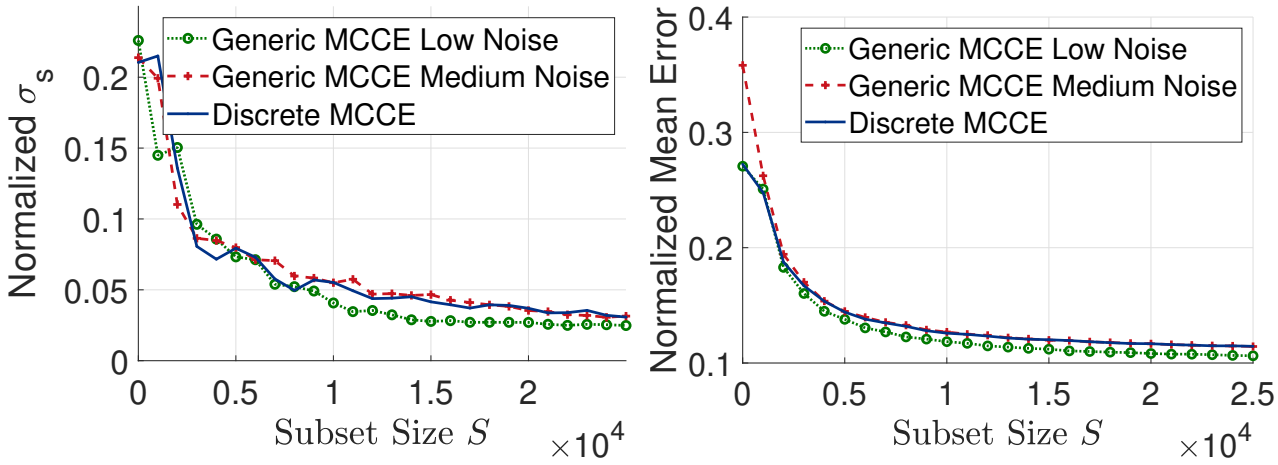
3.3 MCCE Convergence

During development, the convergence of all Monte Carlo operations were tested to quantify the correlation between the error and number of iterations. This ensures that enough iterations are performed at each step to provide a trade-off between estimator accuracy and estimation time (increased iterations typically increases

accuracy, but also increases the computational complexity of the estimator). Specifically, convergence was tested for the estimation of the un-modeled response (e.g. Figure 3.2(a)) and the final channel estimate (e.g. Figure 3.3). Convergence was also recalculated for the evaluation of metrics for the MCCE estimator. The value of the convergence for each scenario was calculated similarly to the method outlined in section 2.10. Each of these tests was performed using *Propagation Channel 2*.

3.3.1 Convergence of the Unmodeled Response

Quantifying the convergence for the unmodeled response helped decide how many samples of the hardware noise distributions were taken and used in the calculation of the overdefined system of linear equations for the un-modeled response (See section 3.1.2). Subsets were sampled from a total of 1×10^5 iterations used in the unmodeled response. Figure 3.2(a) shows the convergence of the estimated unmodeled response for subsets of up to 2.5×10^4 iterations. The convergence was calculated as the standard deviation between unmodeled response estimates, σ_s, for different subset sizes S normalized to actual unmodeled response estimate. The evaluator was configured with a *Propagation Channel 2* and *Phased Array 1*. This plot gives a baseline of how many iterations are required to estimate an unmodeled response with a given level of error. After about 5×10^3 to 1×10^4 iterations, the curve begins to flatten showing that increasing the number of iterations provides minimal gain.



(a) Normalized standard deviation, σ_s.

(b) Normalized mean error.

Figure 3.2 Convergence of subset means for the unmodeled response $\widehat{\mathbf{h}}_{\text{umod}}$.

The accuracy of the un-modeled response was also calculated for a scenario where the un-modeled response had no noise (only a nominal value for each frequency). Figure 3.2(b) shows the mean of the squared error (normalized to the actual response) between the actual un-modeled response at each element and the

estimated un-modeled response for different subset sizes. This accuracy calculation was performed for the MCCE configured with different modeled responses h_{mod}.

Each of these plots allows a selection of the number of iterations for the un-modeled response estimation that balances computational complexity and accuracy. A value of 1×10^4 was chosen for the MCCE configurations used in this work (unless otherwise specified). On both Figure 3.2(a) and Figure 3.2(b) the curves begin to flatten out just after 1×10^4 repeats. The majority of the error and deviation exists before about 5×10^3 iterations, but this value will change based on the configuration of the MCCE and the configuration of the unmodeled response. Therefore, 1×10^4 was chosen to provide buffer for different configurations.

3.3.2 Convergence of the Channel Estimate

After quantifying and selecting a value for the number of iterations for estimating the un-modeled response, convergence testing was also performed with the final channel estimate. This quantified the convergence of the MCCE as a function of the number of values $\hat{\mathbf{y}}_{mc}$ that were generated per estimation. Figure 3.3 shows the convergence (as calculated in section 2.10) of the standard deviation, σ_s, of the channel estimate given different subset sizes S of output Monte Carlo iterations (i.e. number of $\hat{\mathbf{y}}_{mc}$).

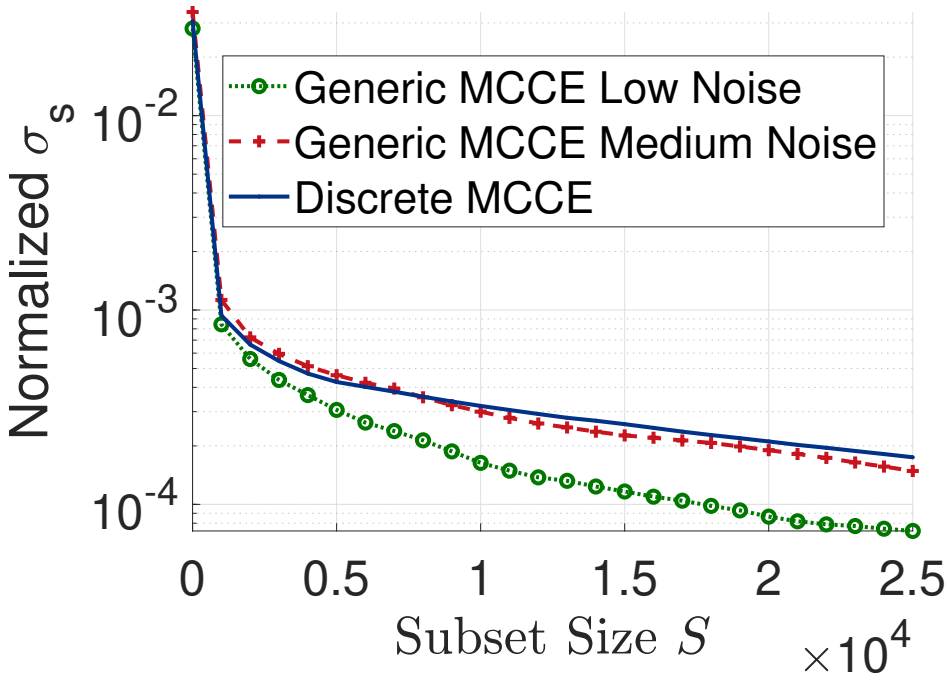


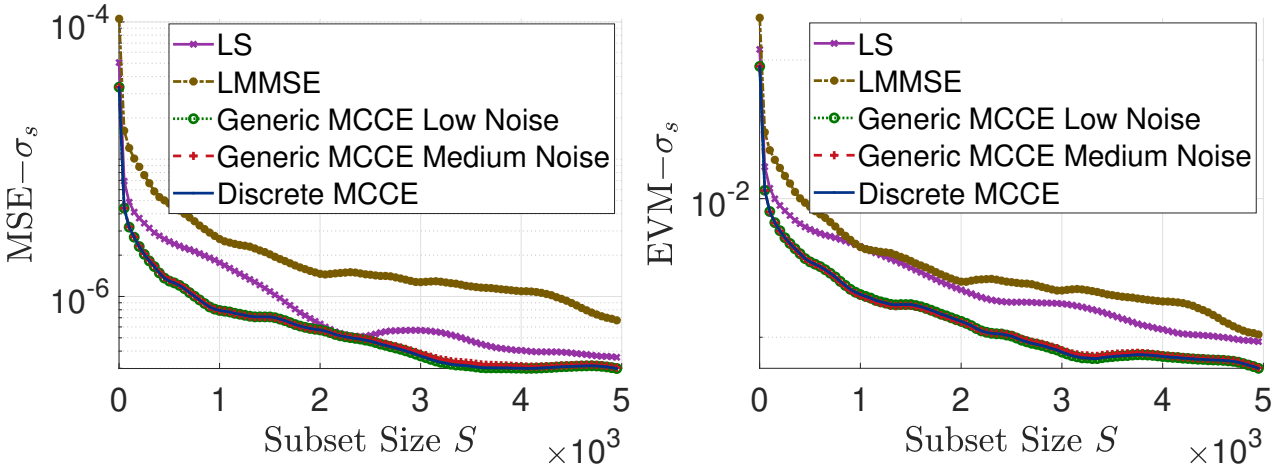
Figure 3.3 Convergence of the standard deviation of subset means for the channel estimation using different MCCE configurations.

This figure shows that the deviation of the estimate quickly converges. With a subset size of only 1000 (1000 iterations, 1000 values of $\hat{\mathbf{y}}_{mc}$) the deviation is reduced by over a factor of 10. Like with the

unmodeled response, this figure allows the selection of a subset size that balances computational complexity and accuracy. Because of the quick convergence in the curve, a subset size of 1000 is used for the calculation of the channel estimate unless otherwise specified.

3.3.3 Estimator Evaluation Convergence Testing

Before the MCCE estimators were evaluated, tests from section 2.10 were also repeated to ensure that enough evaluation repeats were performed to accurately test the estimator's performance. The testing performed here used the exact same configuration for the evaluator as in section 2.10.



(a) Normalized standard deviation of MSE.

(b) Normalized standard deviation of EVM.

Figure 3.4 Convergence of the evaluation with different numbers of repeats (subset sizes S) as given by (2.24) normalized to the nominal values.

Figure 3.4(a) and Figure 3.4(b) shows the convergence of the evaluation of the MSE and EVM respectively for the LS, LMMSE, and MCCE estimators discussed in this section. From these plots, it is seen that both the evaluated MSE and EVM value quickly drop as 1000 repeats is approached. For both metrics, all of the MCCE configurations converge quicker than both the LMMSE and LS estimators. This faster convergence is due to the MCCE reduced variability in estimation over the LS and LMMSE estimators.

The evaluations and comparisons in this work used commonly use a subset size of S between 1000 and 2500 iterations. These values are selected to provide the best tradeoff of error and evaluation time for each test case. In some specific cases, such as those requiring a large amount of transmitted data, subset sizes as small as 100 repeats were used to provide faster evaluation, but at the cost of higher error in the evaluation metrics.

3.4 Symbol Count Effect on Estimator Accuracy

Each of the discussed estimators is able to increase its accuracy by receiving more symbols in a static channel. Estimators like the LS and LMMSE can average across these received symbols to reduce the noise at each pilot tone. This reduced noise in turn leads to a more accurate channel estimation. The downside of increasing the received symbol count is the time it takes to transmit and receive the extra symbols. The MCCE is able to improve estimation by simulating more symbols than were actually received.

While the MCCE simulates more symbols than were received, it still must have a sufficient number of symbols to provide a good estimate of the un-modeled response. If too few symbols are used and the estimate of the un-modeled response is poor, the MCCE can become less accurate than the LS and LMMSE estimators. If enough symbols are used such that the number of received symbols is similar to the number of simulated outputs ($\hat{y}_{mc}$) of the MCCE, the MCCE will not be able to further reduce the noise and will provide no benefit over the LS and LMMSE estimators. By looking at the accuracy of each estimator while varying the number of received symbols, the "Goldilocks Zone" of the MCCE can be found. This is where the number of received symbols will be enough to accurately estimate the un-modeled repsonse, but not too many to where it approaches the number of simulated symbols from the MCCE.

3.4.1 Effect on Estimate of Unmodeled response

The effect of increasing the received symbols on the estimate of the un-modeled response when using a MCCE estimator was also investigated. This was tested by running 10000 iterations to calculate the unmodeled response and averaging across each received symbol. This averaging reduces the noise in the received symbol to provide a better estimate of the unmodeled response. The MCCE was tested using three different array models *Phased Array 1*, *Phased Array 2*, and *Phased Array 4*. Figure 3.5 shows the error of the estimated unmodeled response (as the mean of the square error for each phased array element normalized to the value received by that element) for a varying number of received symbols. This shows that for the case described, the unmodeled response quickly converges to a constant value after about 40 symbols, although the change is minimal between 1 and 40 symbols. This also shows the offset of each of the estimations of the unmodeled response for various MCCE configurations. While the differences in the error of the estimate of the unmodeled response are small, it can clearly be seen that the configurations with the continuous phase shifters (and those that most closely represent the hardware under test) provide a closer estimation than the discretized phase shifters.

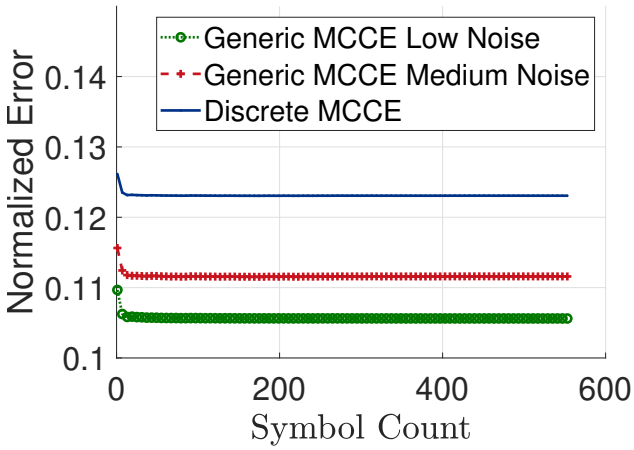
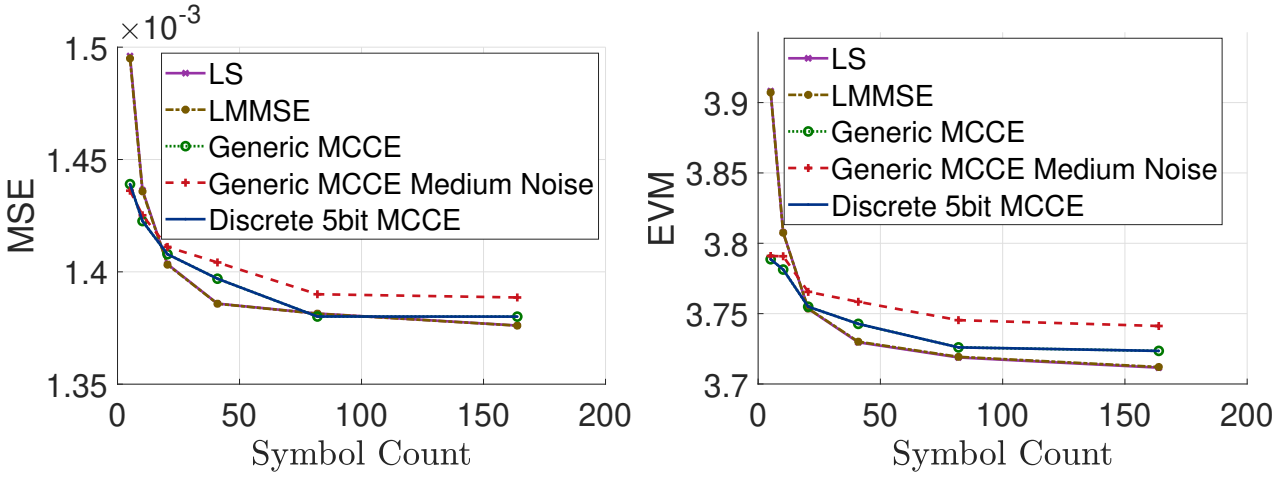


Figure 3.5 Accuracy of the unmodeled response $\hat{\mathbf{h}}_{\mathbf{umod}}$ as a function of symbol count used in estimation.

3.4.2 Effect on Channel Estimation

The effect of the number of received symbols on the total efficacy of multiple estimators was also studied. This compared how each of the MCCE estimator configurations compared against a LS and LMMSE estimator for a varying number of received symbols. As in previous sections, the LS and LMMSE estimators used the multiple symbols by assuming a constant channel and averaging across symbols. For the same test case (*Propagation Channel 2* and *Phased Array 1*), the results of the MSE and EVM of the estimation as a function of symbol count can be seen in Figure 3.6(a) and Figure 3.6(b), respectively. The BER when using 16-QAM is 0 due to the low noise relative to the modulation scheme.

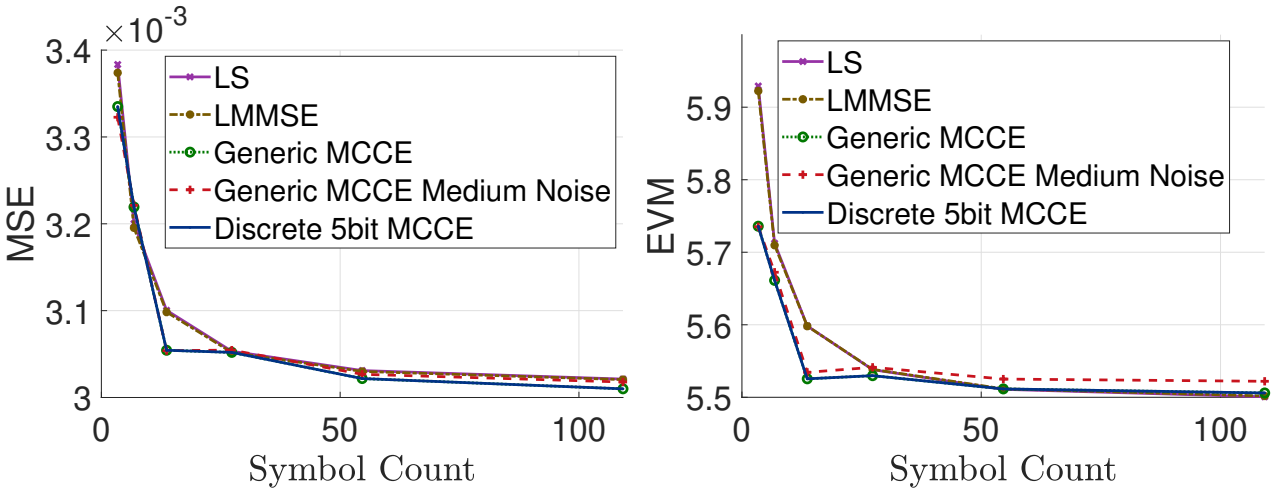


(a) MSE (b) EVM

Figure 3.6 Metrics of each estimator as a function of received symbol count using 16-QAM, *Propagation Channel 2*, and *Phased Array 1*.

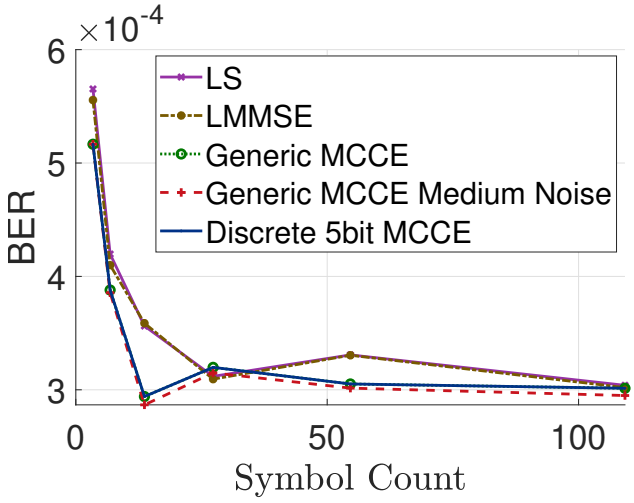
For this configuration of the evaluator, the MCCE outperforms both LS and LMMSE for MSE and EVM for a low number received symbols ($\approx< 20$). After this, the LS and LMMSE estimators surpass the efficacy of the MCCE in all cases. This shows that the use of the MCCE decreases as more symbols are captured. This is because a basic estimator like the LS estimator can accurately estimate the channel through the mean of the received values if provided enough symbols. While this is true, this varies depending on the configuration.

This evaluator was then reconfigured to use *Propagation Channel 2* and *Phased Array 2* to increase the phase noise in the evaluator hardware. The increased noise highlights the efficacy of the estimators as the capacity of a modulation scheme is approached. The MSE and EVM for this configuration can be seen in Figure 3.7(a) and Figure 3.7(b), respectively. The BER for this configuration can be seen in Figure 3.7(c). Unlike when evaluating with *Phased Array 1*, the BER is farther from the noise floor of the evaluator and therefore provides a better metric for comparison. In the case of increased noise, the MCCE configurations provide an improved evaluation for a lower number of symbols (again $\approx< 20$). The LS and LMMSE provide near identical results for MSE and EVM as more symbols are used. Even as the LMMSE and LS estimators converge on the value of MSE and EVM of the MCCE, the BER of the MCCE is still typically lower than that of the LS and LMMSE. This again is due to the lower standard deviation of the estimation from the MCCE. While the MSE and EVM are important, the BER is what directly impacts the throughput of the system and therefore because the BER is reduced, a communications system would be able to increase throughput and stay within a higher order modulation scheme for longer when using the MCCE. It is also important to note that each of the three versions of the MCCE performs very similarly, despite the Medium Noise configuration being closest to the hardware in the evaluator.



(a) MSE

(b) EVM



(c) BER

Figure 3.7 Metrics of each estimator as a function of received symbol count using 64-QAM, *Propagation Channel 2*, and *Phased Array 2*.

3.5 Efficacy of the MCCE Compared to Other Channel Estimators

After the testing of convergence and accuracy of the MCCE alone and testing the effect of received symbols, full evaluations using the evaluator were run to compare the efficacy of the MCCE against the LS and LMMSE estimators. SNR vs BER sweeps were performed to visualize the difference in estimation accuracy between various configurations of MCCE and the other estimators. These arrays were all variations of *Phased Array 1*, *Phased Array 2*, and *Phased Array 3* with different phase and amplitude noise levels. Other results were also gathered to compare how MCCE compared against the other estimators for the static arrays in section 3.2. For example, arrays like the S-Parameter defined array *Phased Array 5* has predefined values that do not change.

The MCCE were configured to use 10000 iterations to calculate $\hat{\mathbf{h}}_{umod}$ and 1000 iterations of $\hat{y}_{mc}$ were generated to calculate $\hat{h}_{MCCE}$. The evaluator was configured to use 6 repeat symbols with 1000 repeat evaluations (unless otherwise specified), QPSK, 16-, 64-, 256-, 1024-, and 4096-QAM modulation schemes with a comb-type pilot arrangement every 10 subcarriers. The evaluator was configured with a variety of different propagation channels and phased array hardware configurations to test operation in varying scenarios.

3.5.1 Comparison of Estimators vs Signal to Noise Ratio

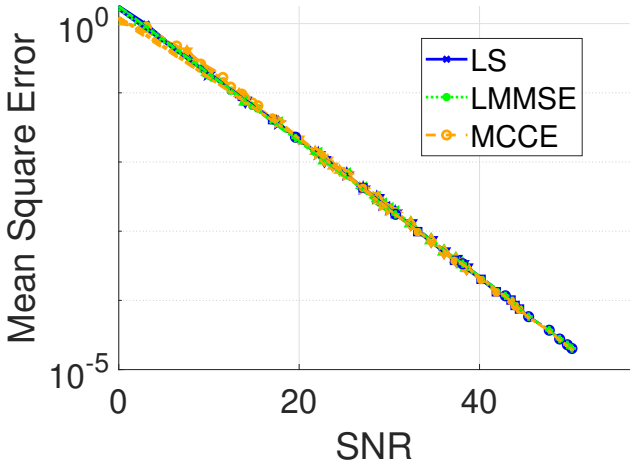
The MCCE was compared against the LS and LMMSE estimator by looking at their output metrics of MSE, EVM, and BER as the SNR of a communications system changes. This provides insight into how the estimators may operate as their environment changes and different SNR values are presented to a communication system. This comparison starts out by generating an array and channel configuration that provides an SNR value close to the desired value. Depending on the modulation, SNRs ranging from 0 to 50 dB were generated. The SNR in each scenario is calculated as the decibel form of the division of the average output magnitude divided by the noise calculated as the difference of the output and the input. This is written as

$$\text{SNR} = 20\log_{10}\frac{E[A_{out}]}{E[(A_{out} - A_{in})]}. \tag{3.6}$$

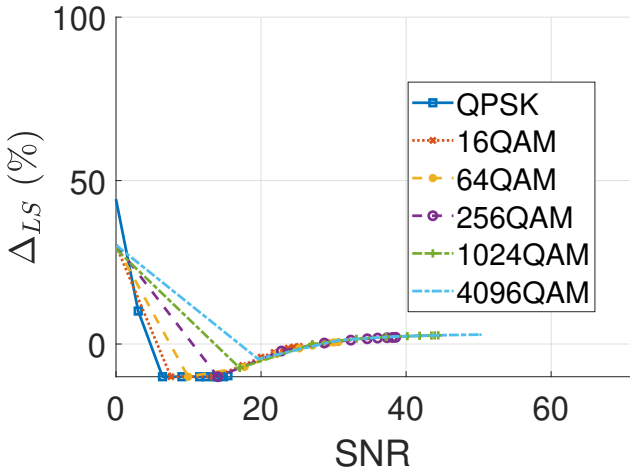
In each of these cases, the LS estimator stays unchanged, new LMMSE auto-correlation matrices are generated and used, and a new MCCE is created to match the SNR of the system. While the LS in this situation requires no changes, the other two take extra steps to provide improved performance.

In the first case, a phased array with continuous phase shifters and unity gain amplifiers was tested (similar to *Phased Array 1*) with the boresight propagation channel *Propagation Channel 1*. The results

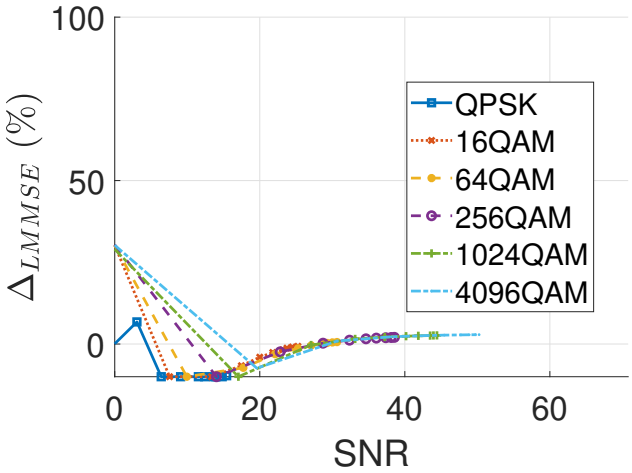
for each of the estimators for MSE, EVM, and BER can be seen in Figure 3.8, Figure 3.9, and Figure 3.10 respectively. Each of these results provides the measured values along with the relative values between MCCE to LS and MCCE to LMMSE.



(a) Mean squared error vs. signal to noise ratio for varying modulations.



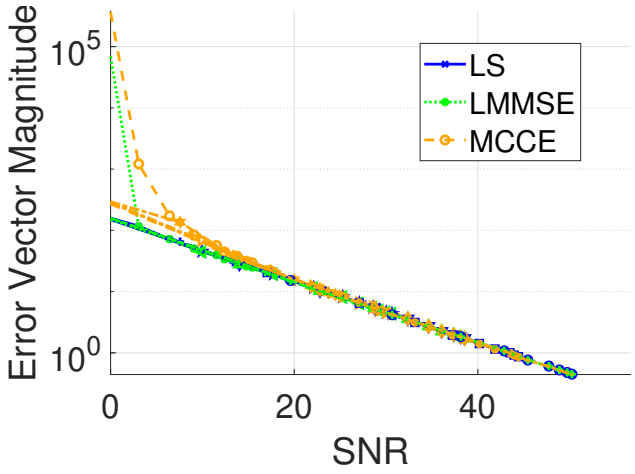
(b) Percent decrease in MSE of MCCE over an LS estimator for varying signal to noise ratios.



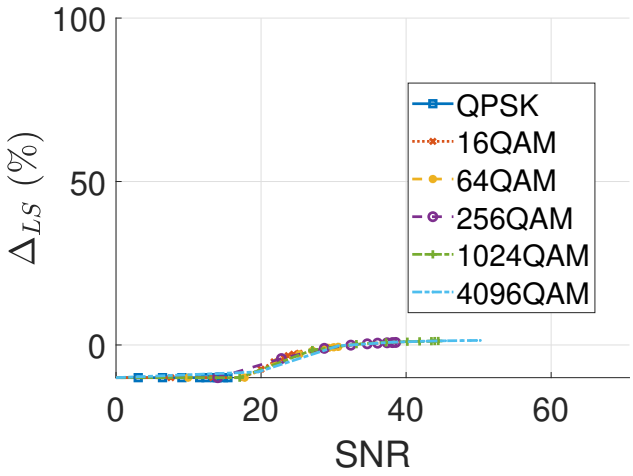
(c) Percent decrease in MSE of MCCE over an LMMSE estimator for varying signal to noise ratios.

Figure 3.8 Comparison of multiple channel estimators for varying signal to noise ratios.

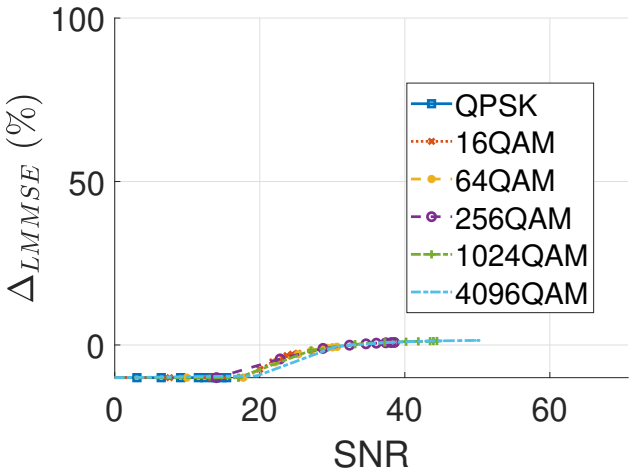
The MSE and EVM comparison in Figure 3.8 and Figure 3.9 show minimal to no improvement when using the MCCE. Because a boresight channel with minimal frequency response is in use, the improvement over the LMMSE and LS estimators is nearly identical. In this case, the correlation matrix used in the LMMSE is approximately equal to the identity matrix it is expected that the LMMSE will approximately equal the LS estimate. By taking a look closer at the standard deviation of these values though for a single point in Table 3.1, it can be seen that the standard deviation for the MCCE is a factor of 10 lower than the LS and LMMSE. This table is created from the values in Figure 3.10(a) at an SNR of approximately 38.6



(a) Error vector magnitude vs. signal to noise ratio for varying modulations.



(b) Percent decrease in EVM of MCCE over an LS estimator for varying signal to noise ratios.

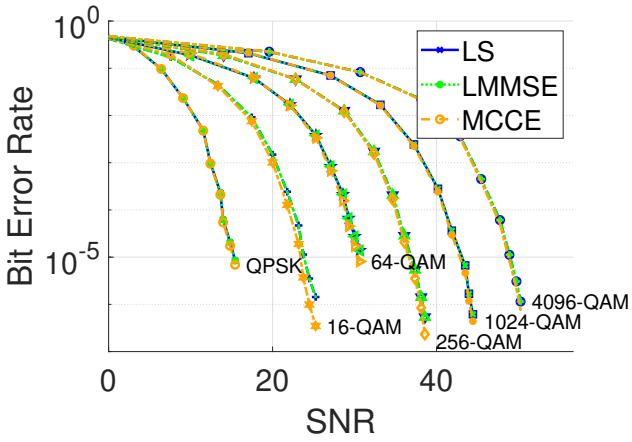


(c) Percent decrease in EVM of MCCE over an LMMSE estimator for varying signal to noise ratios.

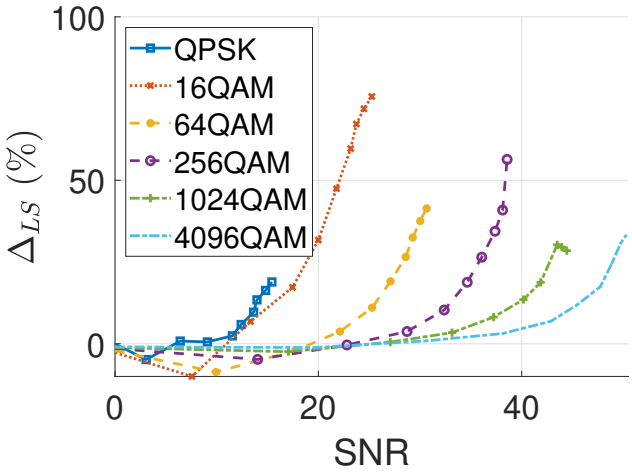
Figure 3.9 Comparison of multiple channel estimators for varying signal to noise ratios.

Table 3.1 Estimator metrics with 256-QAM and 38.6 dB SNR. Results are provided as $Nominal(St.Dev.)$.

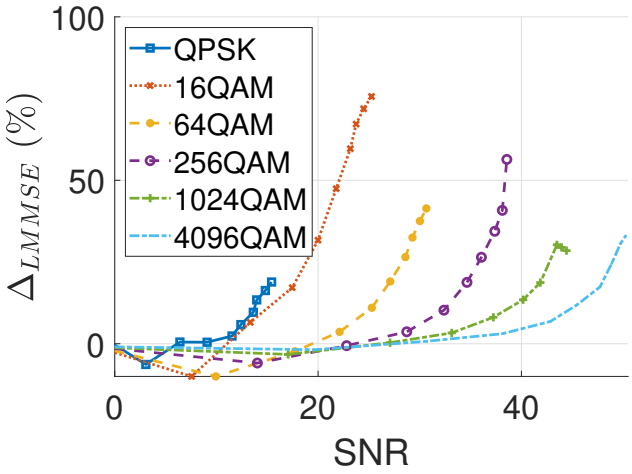
	LS	**LMMSE**	**MCCE**
MSE	0.00029833(1.5628e-05)	0.00029832(1.5625e-05)	0.00029223(1.2602e-06)
EVM	1.7269(0.04479)	1.7268(0.044785)	1.7135(0.0044346)
BER	5.3787e-07(1.0662e-06)	5.3787e-07(1.0662e-06)	2.346e-07(6.6037e-07)



(a) Bit error rate vs. signal to noise ratio for varying modulations.



(b) Percent decrease in bit error rate of MCCE over an LS estimator for varying signal to noise ratios.



(c) Percent decrease in bit error rate of MCCE over an LMMSE estimator for varying signal to noise ratios.

Figure 3.10 Comparison of multiple channel estimators for varying signal to noise ratios.

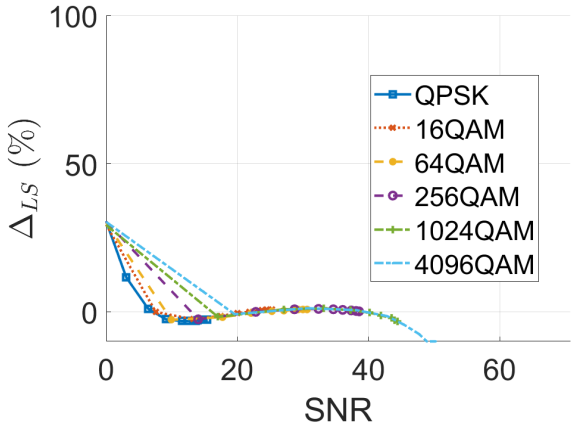
dB with 256-QAM modulation The distribution in MCCE estimation is tighter and has fewer outlier values that are incorrectly demodulated for a sufficiently low modulation scheme and SNR. The tighter distribution translates into a lower BER as seen in Figure 3.10. The reduction in BER is the most important as it directly correlates to higher throughput in a communications system. This comparison highlights how the MCCE can leverage any uncertainties of a system to provide a more precise estimation, reduce the BER, and improve the performance of a communications system. The improvement over the other estimators becomes more noticeable as the SNR is increased, but before it is sufficiently high to produce no bit errors for a given modulation.

3.5.2 MCCE Model Mismatch Effect on Estimator Accuracy

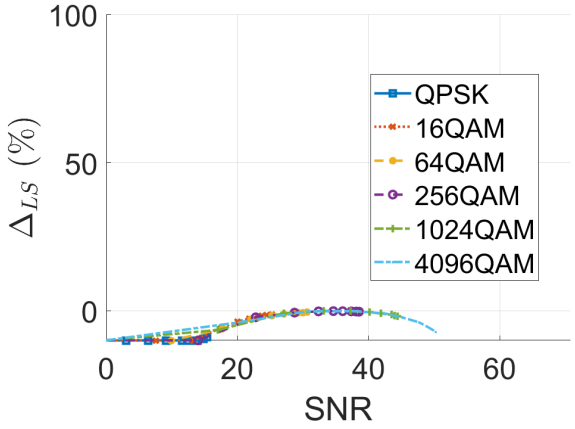
In section 3.5.1 it was demonstrated how the MCCE provides an improved BER by capturing the uncertainties in a communications system. Section 3.5.1 assumed that the models provided to the MCCE were perfect matches to those in the communications system. While this is useful to test for ideal scenarios, real life implementation would never be able to capture with such high accuracy. Further tests were performed to see how the channel estimation from the MCCE was affected by mismatches between the modeled hardware components in the MCCE and the simulated communications system. In these tests, the same SNR sweeps from section 3.5.1 were performed, but the models provided to the MCCE were kept static at a single SNR value.

Figure 3.11 shows how the MCCE performs compared to LS with the MCCE models static at a single SNR at about 28 dB (denoted by the vertical line). In the MSE and EVM plots, the response around the SNR of the models provided to the MCCE is very similar to when the swept SNR results. There are changes though at the upper and lower ends of the plot. Specifically as the SNR becomes higher than what the MCCE models indicate, the MCCE begins to lose accuracy and under-performs the LS estimator. At the lower SNR values, the MCCE almost seems to perform slightly better than before. This is a misleading result though because the extremely high noise at this level makes any sort of estimation nearly impossible. The MCCE though provides a lower standard deviation so as the LS becomes completely inaccurate due to the high noise level, the MCCE will appear to provide better metrics. In reality, 5G systems are typically unable to operate at these noise levels and therefore these estimators will not be operating in these regimes.

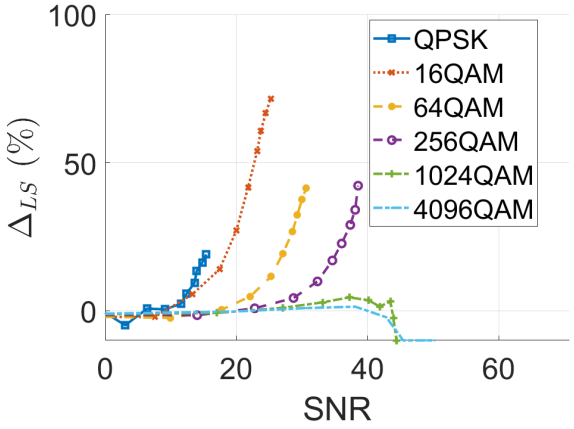
These plots show that even if the models of the MCCE are not perfect (as would be the case in real implementations), it still can outperform LS and LMMSE estimators. If the models stray too far from the actual hardware responses though the MCCE does lose some of its efficacy seen when models match exactly. This shows that the MCCE does provide some capability to handle inaccurate models, although if the models become too inaccurate, it will become less effective than other estimators.



(a) Bit error rate vs. signal to noise ratio for static MCCE models.



(b) Bit error rate vs. signal to noise ratio for static MCCE models.



(c) Bit error rate vs. signal to noise ratio for static MCCE models.

Figure 3.11 Comparison of MCCE with static models with SNR $\approx 28dB$ relative to LS for varying signal to noise ratios. The vertical line denotes the SNR of the models provided to the MCCE.

3.5.3 MCCE Efficacy with Measurement Based Simulations

Section 3.5.1 provides good insight into how the MCCE performs for generic models, but measurement based scenarios were also tested to get a better idea of more real world performance. These scenarios included testing how the LS, LMMSE, and MCCE performed when a system was simulated with the S-Parameter based array, *Phased Array 5*. This also included testing using the measured *Propagation Channel 3*. The combination of these two provide a realistic simulation of how the MCCE may operate for a single hardware configuration and multi-path propagation channel. The results in this section were simulated using the same parameters provided in 3.5. These simulations were performed using 64- and 256-QAM with an 8x8 planar phased array. The number of repeat simulations in each of these cases was reduced to 100 due to the increased complexity of simulation using measurement based hardware models and propagation channels.

The MCCE was first compared to LS and LMMSE estimators in a simulated system using *Propagation Channel 1* coupled with *Phased Array 5*. The LS, LMMSE, and MCCE estimators were each simulated and compared in this configuration. The MCCE used the same *Phased Array 5* models that were being used in the simulation. The results from this configuration can be seen in Table 3.2. From this table it

Table 3.2 Estimator metrics with 64-QAM, *Propagation Channel 1*, and *Phased Array 5*. MCCE models are the same as *Phased Array 5*. Results are provided as $Nominal(St.Dev.)$.

	LS	**LMMSE**	**MCCE (S-Parameter)**
MSE	0.004 (0.00047)	0.004 (0.00046)	0.0038 ($5.3e-05$)
EVM	6.3 (0.36)	6.3 (0.35)	6.2 (0.047)
BER	0.00075 (0.00045)	0.00073 (0.00044)	0.00055 ($6.8e-05$)

can be seen that even when using S-Parameter based hardware models to more accurately simulate a real system, the MCCE still provides an approximately 7.5x decrease in the standard deviation of the EVM. This in turn provides a nearly 25% decrease in the mean BER. Table 3.3 provides the results for the same simulation configuration except using 256-QAM. As seen in the SNR sweeps in section 3.5.1, as the density

Table 3.3 Estimator metrics with 256-QAM, *Propagation Channel 1*, and *Phased Array 5*. MCCE models are the same as *Phased Array 5*. Results are provided as $Nominal(St.Dev.)$.

	LS	**LMMSE**	**MCCE (S-Parameter)**
MSE	0.004 (0.00046)	0.004 (0.00045)	0.0038 ($5.9e-05$)
EVM	6.4 (0.38)	6.3 (0.37)	6.2 (0.055)
BER	0.019 (0.0029)	0.019 (0.0028)	0.018 (0.00048)

of the modulation scheme is increased (e.g. from 64-QAM to 256-QAM) for a static SNR, the performance gain seen when using MCCE is reduced. This same effect is seen between Table 3.2 and Table 3.3 where the tables go from seeing a reduction in BER of $\approx 25\%$ to seeing almost no gain when using the MCCE. It is important to notice here though that it is unlikely a communications system would use 256-QAM in this case because the EVM values in Table 3.3 are almost double the maximum allowable values given in the specification in Table 2.6.

To further increase the realism of the simulation, the simulated boresight *Propagation Channel 1* was then replaced with the measured *Propagation Channel 3*. This test used the same S-Parameter based *Phased Array 5* as the simulation of Table 3.2 and Table 3.3. Because the S-Parameter based *Phased Array 5* is still being used, the models for the MCCE do not change. The results for this simulation using a measured propagation channel and hardware for 64- and 256-QAM can be seen in Table 3.4 and Table 3.5 respectively.

In this scenario, MCCE provides a $\approx 17\%$ gain over the LMMSE and $\approx 20\%$ gain over LS. While the

Table 3.4 Estimator metrics with 64-QAM, *Propagation Channel 3* , and *Phased Array 5*. MCCE models are the same as *Phased Array 5*. Results are provided as $Nominal(St.Dev.)$.

	LS	**LMMSE**	**MCCE (S-Parameter)**
MSE	0.0065 (0.00064)	0.0064 (0.00062)	0.0063 ($7.6e-05$)
EVM	8.1 (0.39)	8.1 (0.38)	8 (0.056)
BER	0.003 (0.00098)	0.0029 (0.00094)	0.0024 (0.00014)

Table 3.5 Estimator metrics with 256-QAM, *Propagation Channel 3* , and *Phased Array 5*. MCCE models are the same as *Phased Array 5*. Results are provided as $Nominal(St.Dev.)$.

	LS	**LMMSE**	**MCCE (S-Parameter)**
MSE	0.0065 (0.00063)	0.0065 (0.0006)	0.0063 ($8e-05$)
EVM	8.1 (0.41)	8.1 (0.4)	8.1 (0.057)
BER	0.035 (0.0034)	0.034 (0.0033)	0.033 (0.00054)

relative gain seen by using the MCCE here is reduced when compared to the last simulation, the EVM in this simulation has been increased. Therefore in a case with hardware and a realistic channel matching the EVM of previous tests, it is still expected to see similar relative reductions in BER. In this case, the EVM is just over the specified limit to use 64-QAM meaning that typically a communications system would actually jump to 16-QAM. This scenario is a good use case for the MCCE because a BER cannot be produced that is comparable to if LS or LMMSE was used with a lower EVM. As in previous simulations, it was seen that as the higher density modulations are used, but SNR is kept constant, the relative reduction in BER between

MCCE compared to LS and LMMSE decreases providing almost no gain at 256-QAM.

3.6 Justification of MCCE Results

The MCCE provides a reduced BER by lowering the standard deviation of the channel estimation. The relation between the standard of the channel estimation and the BER is due to the non-linear nature of demodulation based on thresholding. Figure 3.12 provides a visualization of constellation used for demodulation with the thresholds lines bolded. Each point in the constellation is mapped to the constellation point contained by the box in which it lies. As higher modulation schemes are used (e.g., 64-QAM) the distance between these thresholds, and the size of the bounding boxes, is decreased. Figure 3.13 shows example distributions of channel estimations between the MCCE and an LS estimator for a single point in a constellation. Areas shaded in red are outside the demodulation thresholds. Any part of the distribution in this shaded area will therefore be incorrectly demodulated and increase the BER. In each of these cases, the MCCE and LS distributions have near identical EVM values. The MCCE has a standard deviation about 10 times less than the LS estimator, which is consistent with previous results. For the 16- and 64-QAM modulations, the MCCE has a higher amount of its distribution within the thresholds compared to the LS estimator. More symbols will therefore be correctly demodulated with the MCCE and provide a lower BER. This reduced BER could allow a communications system to maintain a higher-order modulation for a given EVM than in specifications of EVM vs. modulation like [96] by shaping the estimation distribution.

As either SNR increases (which in turn increases EVM) or the threshold boundaries are moved in (i.e., as higher modulation schemes are used) the estimation distributions approach the demodulation thresholds. When the MCCE distribution begins to cross the threshold it quickly begins to under perform the LS because of its low standard deviation. This is shown in the 256- and 1024-QAM modulations of Figure 3.13. In these figures, almost all of the MCCE distribution lies outside of the threshold boundaries and will therefore be incorrectly demodulated while the LS still has much of its distribution within the boundaries. It is also worth noting that the shapes of these distributions (i.e. their mean and standard deviations) are affected by multiple parameters of the channel estimator. For example, the accuracy of the mean of each distribution is highly dependent on the SNR and the number of symbols used in each estimation. The number of symbols used in each estimation also effects the standard deviation of the LS estimator. The standard deviation of the MCCE is heavily affected by the number of Monte Carlo iterations used. Each of these parameters can be adjusted to fit the needs of a specific system, but the MCCE provides the ability to shape the estimation distribution and provide an improved BER without requiring an increased number of received symbols. While other estimators such as the LMMSE can provide improved accuracy and reduce the standard deviation over the LS using frequency correlation, the results have shown in many cases this reduction in standard deviation

is slight. The previous results show the LMMSE is unable to reduce the standard deviation to the levels seen by the MCCE in a typical system. The MCCE is therefore also able to outperform more complex methods such as the LMMSE in the cases tested.

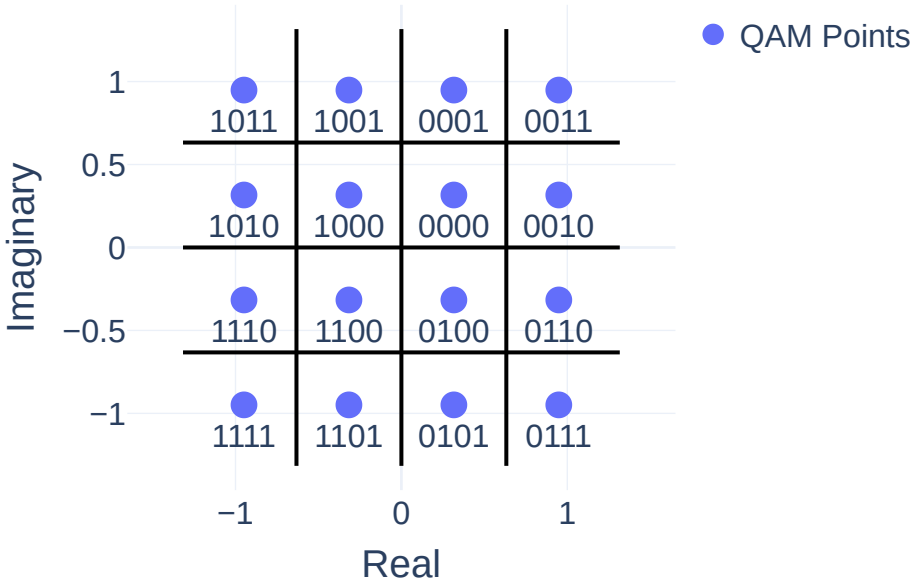
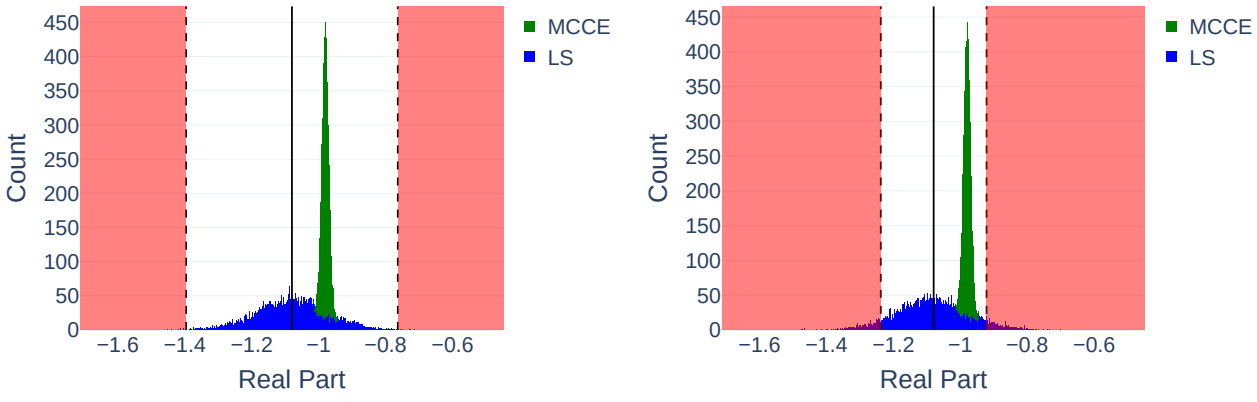


Figure 3.12 Constellation diagram for a 16QAM modulation scheme with bolded thresholds for each QAM point.

3.7 Applications of the MCCE

In section 3.5 it was shown that in many cases, the MCCE outperforms its LS and LMMSE counterparts. The reduction in MSE, EVM, and BER come at the cost of computational complexity as the MCCE requires repeat Monte Carlo simulations to calculate the unmodeled response and the channel estimate. It also assumes that the total response is slowly varying such that multiple symbols can be used to estimate the error distribution. The timeframes required for the symbols needed for the MCCE in section 3.4 can be put into perspective in a 5G application. In 5G systems using 15 kHz subcarrier spacing, each symbol must be transmitted for $\approx 16\ \mu s$. Using the symbol length, capturing 4 and 110 symbols requires $\approx 64\ \mu s$ and $\approx 1760\ \mu s$, respectively. This means that in order to provide the performance discussed here, the channel and nominal hardware responses must be static for this given period of time. This assumption of a static channel is what allows averaging to be performed. Therefore, the MCCE is best in communications systems that can meet the following requirements:



(a) 16-QAM

(b) 64-QAM

(c) 256-QAM

(d) 1024-QAM

Figure 3.13 Distributions of the real part of MCCE and LS estimations from a constellation diagram. These graphics demonstrate how a tighter distribution of estimations can drastically improve BER for sufficiently low noise. Any values in the red shaded area will be demodulated incorrectly. The correct demodulation area is halved as the modulation density is increased.

1. Offload processing to a computational unit such as a co-processor with a base station. This co-processor must be able to handle large amounts of processing to calculate the MCCE estimate in real time.

2. Ensure a slowly varying channel that is nearly static over hundreds of microseconds to milliseconds. This allows multiple symbols to be captured to provide an accurate estimation of the unmodeled response.

3. Provide responses and uncertainties for models of the hardware components in use. Noise distributions and any systematic offsets are captured in the Monte Carlo simulations. These could be obtained from equation based approximations or measurements.

Some possible scenarios that would meet these requirements are as follows:

Application 1 A factory floor has many sensors tracking the operations of different machines. Each of these machines contains a low cost transmitter that sends information back to a single base station with a steerable phased array. While the transmitter for each machine would be low cost due to the large quantity required, the single base station could easily have a co-processor with a hardware acceleration unit. This extra computational capability would allow the base station to calculate multiple estimates using the MCCE for each machine in the factory. Factories with slow moving environments that slowly change make some estimators requiring *a priori* information (like LMMSE) unfeasible. This is a perfect situation for the MCCE as the hardware components do not change, the environment is slowly changing, and extra computational power is available.

Application 2 The "last mile" of connection to houses in a neighborhood uses 5G mmWave communications to provide broadband internet without the need to bury cables. A phased array base station that provides internet for multiple houses is outside on a telephone pole. Each house also contains a device to communicate with the base station. This device may be moved around the house, and furniture within the house may be moved, affecting the response of the propagation channel. Outdoor changes may also occur such as changes in weather and cars parked on the street. These environmental changes prevent estimators like LMMSE from being feasible in the system. Adding extra compute power to the user devices would again allow the MCCE to excel in this situation. The environment is slowly varying, hardware could be pre-characterized for each user device before being sent to the user, and extra computational power could be added to the user device to provide the capability for computing the MCCE.

In scenarios where these requirements are met, it would likely be best to utilize the MCCE alongside a more basic estimator such as LS. The MCCE would be used when the system approaches the capacity of a modulation scheme (i.e. the high EVM begins to produce bit errors), allowing the system to stay in higher modulation schemes (e.g. 64QAM compared to 16QAM) for a longer period of time. In situations where the system error is well within the modulation scheme, the LS estimator could be used. This hybrid approach would take advantage of the reduced variation in estimation (and thus reduced bit errors) provided by the MCCE, while also reducing computational complexity when the MCCE is not required. As discussed in section 3.5, this hybrid approach could be evaluated and adjusted before deployment using a tool like in chapter 2 to provide optimal efficiency and throughput in a communications system.

CHAPTER 4

CONCLUSIONS

This dissertation has presented two new contributions to the field of channel estimation as it relates to 5G mmWave communication systems. These contributions are an improved evaluator for channel estimators and a novel Monte Carlo augmented channel estimator.

The evaluator (chapter 2) developed allows for a consistent comparison of multiple different channel estimators in a variety of hardware and propagation channel configurations. This evaluator was verified through analytical methods and comparing against commercial software and shown to correctly evaluate a variety of channel estimators in different communications system configurations. This included various simulated phased array hardware configurations with both simulated and measured propagation channels. The verified software was then used to compare a number of different estimators, including the new estimator developed in this dissertation. This in turn allowed for the finding of configurations where different channel estimators outperformed others. These estimators could then be used in tandem in a base station to balance between the computational complexity of more accurate estimators with the speed of more simple estimators when the increased accuracy is not needed. The evaluator is also an important tool to test the efficacy of channel estimators for specific 5G systems before those systems are deployed. By comparing different channel estimators for specific configurations, engineers can save time and money by performing tests before deployment. For 5G this is especially important as a large number of mmWave base stations must be deployed all over the world to provide good coverage due to the high path loss. The channel estimator evaluator presented in this dissertation provides an improvement on previous evaluators and frameworks by providing the speed and scalability necessary for the large-scale systems used in 5G while still including uncertainties that occur in mmWave systems. The work covering this evaluator was published in [99].

The developed Monte Carlo augmented channel estimator (chapter 3) provides an improved channel estimation and communication system throughput over previously developed channel estimators by leveraging Monte Carlo methods and *a priori* information about communications system hardware. The MCCE was shown to be capable of providing a reduced MSE and EVM along with a reduced standard deviation over repeat estimations. These reduced metrics and lower variation therefore can drastically reduce the BER, which in turn increases the throughput of a communications system. This result though is dependent on the configuration and propagation environment of the communication system. The decrease in BER is especially noticeable as the error boundaries of a given modulation scheme are approached. As EVM is reduced (typically through a reduction of the communication system error) and the boundaries of the modulation scheme

become further from the received data, the MCCE becomes less effective while staying computationally complex. While the efficacy of the MCCE is situation dependent, it could be used in practice (possibly alongside less computationally complex estimators) to reduce bit errors and improve throughput in communications systems with varying hardware and in different propagation environments. This hybrid approach could be used to reduce bit errors and increase throughput when the MCCE is most efficient, but allow for faster estimation when the MCCE is not required. The work regarding the MCCE was published in [100*].

4.1 Future Work

Both the evaluator and MCCE presented in this dissertation continue to be improved upon to provide more accurate evaluation and channel estimation.

4.1.1 Channel Estimator Evaluator

Continued work on the channel estimator evaluator in chapter 2 includes the definition of more hardware components found in a typical communications system. By extending the evaluator to include more hardware components, the simulation of the hardware will become more accurate. Increasing the accuracy of the modeled responses and uncertainties of the hardware components when compared to their real life counterparts will further increase the accuracy of the channel estimator evaluation.

New components would include digital to analog and analog to digital converters (DAC/ADC). By including the digital to analog interfaces, effects from the discretization of analog data on channel estimation can be characterized. This holds true especially for systems using ADCs with only a few bits to reduce the power consumption in large arrays [27].

Further component extensions would include components such as IQ modulators and demodulators. The addition of IQ modulators and demodulators would allow for the integration of other common uncertainties in communications such as IQ imbalance and quadrature error [101]. This implementation requires introducing time domain simulations into the evaluator. This could be implemented by creating a hybrid simulation that converts from frequency to time domain (or vice versa) for different hardware components. This hybrid time domain would also allow for the introduction of uncertainty sources such as phase noise and frequency offset due to local oscillator (LO) errors. Channel estimators geared toward correcting these uncertainties such as in [97] could therefore be better characterized.

4.1.2 Monte Carlo Augmented Channel Estimator

Based on the data from experiments to this point, it is expected that by introducing more accurate hardware components to the MCCE, the estimation will become more accurate. Therefore, the MCCE could also gain from using hardware components with improved accuracy like the IQ modulation/demodulation

and ADC components discussed in section 4.1.1. While further testing would be required, the MCCE may benefit by including improved hardware components in its modeled response (i.e. h_{mod}). This could allow the MCCE better correct for the bias that each component adds to the system through its responses and uncertainties.

Beyond the implementation of more accurate models into the MCCE, further exploration could be done into the efficacy implementing other forms of Monte Carlo augmented channel estimators. As described in 3.1.5, an alternate approach could be an MCCE that augments the LMMSE estimator to require *a priori* knowledge that is more readily available to a typical communications system. This implementation would allow the usage of a LMMSE estimator without needing to know the entire response from the transmitter to receiver in a communications system. Another possible implementation could augment machine learning based estimators [32, 33] that are able to supplement their typical training data with simulations, similar to work performed in other non-communications fields [102]. This reduces the requirement for the pre-collection of large training data sets typically used with machine learning based estimators.

While the MCCE presented here was thoroughly characterized in a virtual environment using the estimator evaluator, this evaluation is unable to capture all the effects present in a real communications system. It therefore would be beneficial to further investigate the efficacy of the MCCE in a real communications system. This measurement would require a 5G device with sufficient computational power to calculate the MCCE while also providing direct access to uncorrected received data. After implementing the MCCE on this custom device, the device could then be connected to and communicate with a standard 5G base station in different propagation channels. Repeat measurements of MSE, EVM, and BER could be performed when communicating with a base station using a MCCE, LS, and LMMSE channel estimator. This test would provide output metrics to further verify the efficacy of the MCCE.

REFERENCES

[1] R. G. Fellers. Millimeter waves and their applications. *Electrical Engineering*, 75(10):914–917, 1956. ISSN: 2376-7804. DOI: 10.1109/EE.1956.6442193.

[2] IEEE Computer Society. IEEE Standard for Information Technology–Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Enhanced Throughput for Operation in License-exempt Bands above 45 GHz. *IEEE Std 802.11ay-2021 (Amendment to IEEE Std 802.11-2020 as amendment by IEEE Std 802.11ax-2021)*:1–768, 2021. DOI: 10.1109/IEEESTD.2021.9502046.

[3] 3GPP. *5G; NR; Overall Description*. Technical Specification 38.3. ETSI, Oct. 2019. URL: https://www.etsi.org/deliver/etsi_ts/138300_138399/138300/15.07.00_60/ts_138300v150700p.pdf (visited on 12/10/2019).

[4] Robert W. Chang. Synthesis of Band-Limited Orthogonal Signals for Multichannel Data Transmission. en. *Bell System Technical Journal*, 45(10):1775–1796, 1966. ISSN: 1538-7305. DOI: 10.1002/j.1538-7305.1966.tb02435.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1966.tb02435.x (visited on 12/19/2019).

[5] B. Saltzberg. Performance of an Efficient Parallel Data Transmission System. *IEEE Transactions on Communication Technology*, 15(6):805–811, Dec. 1967. ISSN: 2162-2175. DOI: 10.1109/TCOM.1967.1089674.

[6] Ali Fatih Demir, Mohamed Elkourdi, Mostafa Ibrahim, and Huseyin Arslan. Waveform Design for 5G and Beyond. *arXiv:1902.05999 [eess]*:51–76, 2018. arXiv: 1902.05999. DOI: 10.1002/9781119333142.ch2.

[7] 3GPP. *5G; NR; Physical Channels and Modulation*. English. Technical Specification 38.211. ETSI, July 2018, 98. URL: https://www.etsi.org/deliver/etsi_ts/138200_138299/138211/15.02.00_60/ts_138211v150200p.pdf (visited on 12/10/2019).

[8] Andrea Goldsmith. *Wireless Communications*. English. 1 edition. Cambridge ; New York: Cambridge University Press, Aug. 2005. ISBN: 978-0-521-83716-3. (Visited on 02/04/2020).

[9] David Tse and Pramod Viswanath. *Fundamentals of Wireless Communication*. en. 1st ed. Cambridge University Press, May 2005. ISBN: 978-0-521-84527-4 978-0-511-80721-3. DOI: 10.1017/CBO9780511807213. URL: https://www.cambridge.org/core/product/identifier/9780511807213/type/book (visited on 02/13/2020).

[10] 3GPP. *5G; NR; Physical layer procedures for data*. Technical Specification 38.214. ETSI, Oct. 2018. URL: https://www.etsi.org/deliver/etsi_ts/138200_138299/138214/15.03.00_60/ts_138214v150300p.pdf (visited on 12/19/2019).

[11] Federal Communications Commission. *Auction 102: Spectrum Frontiers – 24 GHz*. en. 2020. URL: https://www.fcc.gov/auction/102/factsheet (visited on 02/04/2020).

[12] Federal Communications Commission. *Auction 101: Spectrum Frontiers – 28 GHz*. en. 2020. URL: https://www.fcc.gov/auction/101/factsheet (visited on 02/04/2020).

[13] Federal Communications Commission. *Auction 103: Spectrum Frontiers – Upper 37 GHz, 39 GHz, and 47 GHz*. en. 2020. URL: https://www.fcc.gov/auction/103/factsheet (visited on 02/04/2020).

[14] Theodore S. Rappaport, Shu Sun, Rimma Mayzus, Hang Zhao, Yaniv Azar, Kevin Wang, George N. Wong, Jocelyn K. Schulz, Mathew Samimi, and Felix Gutierrez. Millimeter Wave Mobile Communications for 5G Cellular: It Will Work! *IEEE Access*, 1:335–349, 2013. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2013.2260813.

[15] Mingyang Lei, Jianhua Zhang, Tian Lei, and Detao Du. 28-GHz indoor channel measurements and analysis of propagation characteristics. In: *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*. 2014, 208–212. DOI: 10.1109/PIMRC.2014.7136161.

[16] Alec Weiss, Jeanne Quimby, Rodney Leonhardt, Robert Jones, Josh Kast, Benjamin Jamroz, Peter Vouras, Dylan Williams, and Kate A. Remley. Millimeter-Wave High Multipath Channel Measurements. en. In: *2020 XXXIIIrd General Assembly and Scientific Symposium of the International Union of Radio Science*. Rome, Italy: IEEE, Aug. 2020. ISBN: 978-94-6396-800-3. DOI: 10.23919/URSIGASS49373.2020.9232158. URL: https://ieeexplore.ieee.org/document/9232158/ (visited on 03/24/2021).

[17] Antonio Possolo. Simple Guide for Evaluating and Expressing the Uncertainty of NIST Measurement Results, (NIST TN 1900):NIST TN 1900, Oct. 2015. DOI: 10.6028/NIST.TN.1900. URL: https://nvlpubs.nist.gov/nistpubs/TechnicalNotes/NIST.TN.1900.pdf (visited on 06/01/2021).

[18] Kim Hassett. Phased Array Antenna Calibration Measurement Techniques and Methods. *NSI-MI Technologies*, 2016. URL: https://www.nsi-mi.com/images/Technical_Papers/2016/EUCAP2016_KH_Phased_Array_Antenna_Calibration_Measurement.pdf.

[19] J. B. Johnson. Thermal Agitation of Electricity in Conductors. *Physical Review*, 32(1):97–109, July 1, 1928. ISSN: 0031-899X. DOI: 10.1103/PhysRev.32.97. URL: https://link.aps.org/doi/10.1103/PhysRev.32.97 (visited on 05/12/2021).

[20] Mattia Rebato, Marco Mezzavilla, Sundeep Rangan, Federico Boccardi, and Michele Zorzi. Understanding Noise and Interference Regimes in 5G Millimeter-Wave Cellular Networks. *arXiv:1604.05622 [cs, math]*, Apr. 19, 2016. arXiv: 1604.05622. URL: http://arxiv.org/abs/1604.05622 (visited on 02/25/2020).

[21] Jan-Jaap van de Beek, Ove Edfors, Magnus Sandell, Sarah Kate Wilson, and Per Ola Borjesson. On channel estimation in OFDM systems. In: 1995 IEEE 45th Vehicular Technology Conference. Countdown to the Wireless Twenty-First Century. Vol. 2. ISSN: 1090-3038. July 1995, 815–819. DOI: 10.1109/VETEC.1995.504981.

[22] S. Coleri, M. Ergen, A. Puri, and A. Bahai. Channel estimation techniques based on pilot arrangement in OFDM systems. *IEEE Transactions on Broadcasting*, 48(3):223–229, Sept. 2002. ISSN: 0018-9316. DOI: 10.1109/TBC.2002.804034. URL: http://ieeexplore.ieee.org/document/1033876/ (visited on 09/29/2019).

[23] Ove Edfors, Magnus Sandell, Jan-Jaap van de Beek, Sarah Kate Wilson, and Per Ola Borjesson. OFDM channel estimation by singular value decomposition. In: *Proceedings of Vehicular Technology Conference*. Vehicular Technology Conference. Vol. 2. ISSN: 1090-3038. Apr. 1996, 923–927. DOI: 10.1109/VETEC.1996.501446.

[24] Meng-Han Hsieh and Che-Ho Wei. Channel estimation for OFDM systems based on comb-type pilot arrangement in frequency selective fading channels. *IEEE Transactions on Consumer Electronics*, 44(1):217–225, Feb. 1998. ISSN: 1558-4127. DOI: 10.1109/30.663750.

[25] Alan V Oppenheim and George C Verghese. Signals, Systems, and Inference — Class Notes for 6.011: Introduction to Communication, Control and Signal Processing Spring 2010, 2010.

[26] Jan-Jaap van de Beek and Sarah Kate. Analysis of DFT-based channel estimators for OFDM, 1995.

[27] Kais Hassan, Mohammad Masarra, Marie Zwingelstein, and Iyad Dayoub. Channel Estimation Techniques for Millimeter-Wave Communication Systems: Achievements and Challenges. *IEEE Open*

Journal of the Communications Society, 1:1336–1363, 2020. ISSN: 2644-125X. DOI: 10.1109/OJCOMS.2020.3015394. URL: https://ieeexplore.ieee.org/document/9165822/ (visited on 02/23/2021).

[28] Wonil Roh, Ji-Yun Seol, Jeongho Park, Byunghwan Lee, Jaekon Lee, Yungsoo Kim, Jaeweon Cho, Kyungwhoon Cheun, and Farshid Aryanfar. Millimeter-wave beamforming as an enabling technology for 5G cellular communications: theoretical feasibility and prototype results. *IEEE Communications Magazine*, 52(2):106–113, Feb. 2014. ISSN: 0163-6804. DOI: 10.1109/MCOM.2014.6736750. URL: http://ieeexplore.ieee.org/document/6736750/ (visited on 03/24/2021).

[29] Jaekyun Moon, Hui Jin, Taehyun Jeon, and Sok-Kyu Lee. Channel estimation for MIMO-OFDM systems employing spatial multiplexing. In: *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004.* IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004. Vol. 5. ISSN: 1090-3038. Sept. 2004, 3649–3654 Vol. 5. DOI: 10.1109/VETECF.2004.1404745.

[30] Evangelos Vlachos, George C. Alexandropoulos, and John Thompson. Wideband MIMO Channel Estimation for Hybrid Beamforming Millimeter Wave Systems via Random Spatial Sampling. *IEEE Journal of Selected Topics in Signal Processing*, 13(5):1136–1150, Sept. 2019. ISSN: 1941-0484. DOI: 10.1109/JSTSP.2019.2937633.

[31] Hongji Huang, Song Guo, Guan Gui, Zhen Yang, Jianhua Zhang, Hikmet Sari, and Fumiyuki Adachi. Deep Learning for Physical-Layer 5G Wireless Techniques: Opportunities, Challenges and Solutions. *arXiv:1904.09673 [cs, eess]*, Apr. 21, 2019. arXiv: 1904.09673. URL: http://arxiv.org/abs/1904.09673 (visited on 03/24/2021).

[32] Wafi Danesh, Chenyuan Zhao, Bryant T. Wysocki, Michael J. Medley, Ngwe N. Thawdar, and Yang Yi. Channel estimation in wireless OFDM systems using reservoir computing. In: *2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*. 2015 IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA). Verona, NY, USA: IEEE, May 2015, 1–5. ISBN: 978-1-4673-7557-3. DOI: 10.1109/CISDA.2015.7208638. URL: http://ieeexplore.ieee.org/document/7208638/ (visited on 01/28/2020).

[33] Chia-Hsin Cheng, Yung-Fa Huang, Hsing-Chung Chen, and Tsung-Yu Yao. Neural Network-Based Estimation for OFDM Channels. In: *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. 2015 IEEE 29th International Conference on Advanced Information Networking and Applications. ISSN: 1550-445X, 2332-5658. Mar. 2015, 600–604. DOI: 10.1109/AINA.2015.242.

[34] Pallaviram Sure and Chandra Mohan Bhuma. A survey on OFDM channel estimation techniques based on denoising strategies. *Engineering Science and Technology, an International Journal*, 20(2):629–636, Apr. 1, 2017. ISSN: 2215-0986. DOI: 10.1016/j.jestch.2016.09.011. URL: http://www.sciencedirect.com/science/article/pii/S2215098616304542 (visited on 12/19/2019).

[35] Christian R. Berger, Shengli Zhou, Weian Chen, and Peter Willett. Sparse channel estimation for OFDM: Over-complete dictionaries and super-resolution. In: *2009 IEEE 10th Workshop on Signal Processing Advances in Wireless Communications*. 2009 IEEE 10th Workshop on Signal Processing Advances in Wireless Communications (SPAWC). Perugia, Italy: IEEE, June 2009, 196–200. ISBN: 978-1-4244-3695-8. DOI: 10.1109/SPAWC.2009.5161774. URL: http://ieeexplore.ieee.org/document/5161774/ (visited on 01/10/2020).

[36] Waheed U. Bajwa, Jarvis Haupt, Akbar M. Sayeed, and Robert Nowak. Compressed Channel Sensing: A New Approach to Estimating Sparse Multipath Channels. *Proceedings of the IEEE*, 98(6):1058–1076, June 2010. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2010.2042415. URL: http://ieeexplore.ieee.org/document/5454399/ (visited on 02/23/2020).

[37] Athar Waseem, Aqdas Naveed, Sardar Ali, Muhammad Arshad, Haris Anis, and Ijaz Mansoor Qureshi. Compressive Sensing Based Channel Estimation for Massive MIMO Communication Systems. *Wireless Communications and Mobile Computing*, 2019:1–15, May 27, 2019. ISSN: 1530-8669, 1530-8677. DOI: 10.1155/2019/6374764. URL: https://www.hindawi.com/journals/wcmc/2019/6374764/ (visited on 01/10/2020).

[38] Paul H. Moose. A technique for orthogonal frequency division multiplexing frequency offset correction. *IEEE Transactions on Communications*, 42(10):2908–2914, Oct. 1994. ISSN: 1558-0857. DOI: 10.1109/26.328961.

[39] Fan Wu and Mosa Ali Abu-Rgheff. FFT-based frequency offset estimation in OFDM systems. In: *Melecon 2010 - 2010 15th IEEE Mediterranean Electrotechnical Conference*. ISSN: 2158-8481. Apr. 2010, 1326–1331. DOI: 10.1109/MELCON.2010.5475975.

[40] Yanxiang Jiang, Xiqi Gao, and Xiaohu You. Frequency Offset Estimation for OFDM Systems with a Novel Frequency Domain Training Sequence. *arXiv:1703.07089 [cs, math]*, Mar. 2017. arXiv: 1703.07089. URL: http://arxiv.org/abs/1703.07089 (visited on 02/04/2020).

[41] Keysight. *PathWave Advanced Design System*. Keysight. Section: Article Section. 2021. URL: https://www.keysight.com/us/en/products/software/pathwave-design-software/pathwave-advanced-design-system.html (visited on 03/04/2021).

[42] *Ansys HFSS | Ansys Electronics*. 2021. URL: https://www.ansys.com/products/electronics/ansys-hfss (visited on 03/24/2021).

[43] Shihao Ju, Ojas Kanhere, Yunchou Xing, and Theodore S. Rappaport. A Millimeter-Wave Channel Simulator NYUSIM with Spatial Consistency and Human Blockage. *arXiv:1908.09762 [cs, eess, math]*, Aug. 26, 2019. arXiv: 1908.09762. URL: http://arxiv.org/abs/1908.09762 (visited on 03/24/2021).

[44] Marco Mezzavilla, Menglei Zhang, Michele Polese, Russell Ford, Sourjya Dutta, Sundeep Rangan, and Michele Zorzi. End-to-End Simulation of 5G mmWave Networks. *IEEE Communications Surveys & Tutorials*, 20(3):2237–2263, 2018. ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2018.2828880. arXiv: 1705.02882. URL: http://arxiv.org/abs/1705.02882 (visited on 01/08/2020).

[45] Stefan Pratschner, Bashar Tahir, Ljiljana Marijanovic, Mariam Mussbah, Kiril Kirev, Ronald Nissel, Stefan Schwarz, and Markus Rupp. Versatile mobile communications simulation: the Vienna 5G Link Level Simulator. *EURASIP Journal on Wireless Communications and Networking*, 2018(1):226, Dec. 2018. ISSN: 1687-1499. DOI: 10.1186/s13638-018-1239-6. URL: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-018-1239-6 (visited on 02/25/2020).

[46] A. Dowler, A. Doufexi, and A. Nix. Performance evaluation of channel estimation techniques for a mobile fourth generation wide area OFDM system. In: *Proceedings IEEE 56th Vehicular Technology Conference*. 2002 IEEE 56th Vehicular Technology Conference. Vol. 4. Vancouver, BC, Canada: IEEE, 2002, 2036–2040. ISBN: 978-0-7803-7467-6. DOI: 10.1109/VETECF.2002.1040576. URL: http://ieeexplore.ieee.org/document/1040576/ (visited on 07/13/2021).

[47] Michael Meidlinger and Qi Wang. Performance Evaluation of LTE Advanced Downlink Channel Estimators. In: *The 19th International Conference on Systems, Signals and Image Processing*. IWSSIP. Vienna, Austria, Apr. 11, 2012, 4.

[48] Robert J. Achatz. Modeling and simulation of a OFDM radio link. In: *Proceedings of 1997 Wireless Communications Conference*. ISSN: null. Aug. 1997, 234–239. DOI: 10.1109/WCC.1997.622285.

[49] Mary Ann Ingram and Guillermo Cacho Acosta. OFDM Simulation Using Matlab. In: 2000.

[50] Paul Guanming Lin. OFDM Simulation in MATLAB. en:59, 2010.

[51] Xiurong Bao. Matlab simulation and performance analysis of OFDM system. In: *2012 International Conference on Computer Science and Information Processing (CSIP)*. ISSN: null. Aug. 2012, 1423–1426. DOI: 10.1109/CSIP.2012.6309131.

[52] Maximilian Matthe. *Python OFDM Example - DSPIllustrations.com*. 2019. URL: https://dspillustrations.com/pages/posts/misc/python-ofdm-example.html (visited on 12/29/2019).

[53] Jinghao Shi. *OpenOFDM 1.0 documentation*. 2017. URL: https://openofdm.readthedocs.io/en/latest/overview.html (visited on 02/04/2020).

[54] Tomas Dominguez-Bolano, Jose Rodriguez-Pineiro, Jose A. Garcia-Naya, and Luis Castedo. The GTEC 5G link-level simulator. In: *2016 1st International Workshop on Link- and System Level Simulations (IWSLS)*. ISSN: null. July 2016, 1–6. DOI: 10.1109/IWSLS.2016.7801585.

[55] Teemu Nyländen, Janne Janhunen, Olli Silvén, and Markku Juntti. A GPU implementation for two MIMO-OFDM detectors. In: *Modeling and Simulation 2010 International Conference on Embedded Computer Systems: Architectures*. ISSN: null. July 2010, 293–300. DOI: 10.1109/ICSAMOS.2010.5642054.

[56] Bhargav Gokalgandhi, Christina Segerholm, Nilanjan Paul, and Ivan Seskar. Accelerating Channel Estimation and Demodulation of Uplink OFDM symbols for Large Scale Antenna Systems using GPU. *arXiv:1901.07499 [cs]*, Jan. 2019. arXiv: 1901.07499. URL: http://arxiv.org/abs/1901.07499 (visited on 09/29/2019).

[57] nsnam. *ns-3*. en. 2020. URL: nsam.org (visited on 01/29/2020).

[58] Thierry Chonavel. *Statistical Signal Processing*. Red. by Michael J. Grimble and Michael A. Johnson. Advanced Textbooks in Control and Signal Processing. London: Springer London, 2002. ISBN: 978-1-85233-385-0 978-1-4471-0139-0. DOI: 10.1007/978-1-4471-0139-0. URL: http://link.springer.com/10.1007/978-1-4471-0139-0 (visited on 07/14/2020).

[59] Zhou Feng, Ji Rui, Sun Jing-Lu, and Zhou Xin. Analysis on the Definition Consistency Problem of EVM Measurement and its Solution. *IEEE Transactions on Instrumentation and Measurement*, 69(2):528–532, Feb. 2020. Conference Name: IEEE Transactions on Instrumentation and Measurement. ISSN: 1557-9662. DOI: 10.1109/TIM.2019.2901561.

[60] Michael D McKinley, Kate A Remley, Maciej Myslinski, J Stevenson Kenney, and Bart Nauwelaers. EVM Calculation for Broadband Modulated Signals:8, 2004.

[61] Keysight Technologies. *How Do I Measure the Bit Error Rate (BER) to a Given Confidence Level on the J-BERT M8020A and the M8040A High-Performance BERT?* 2020. URL: https://www.keysight.com/main/editorial.jspx?ckey=1481106&id=1481106&nid=-11143.0.00&lc=eng&cc=US (visited on 02/04/2020).

[62] Alec Weiss and Atef Elsherbeni. Computational Performance of MATLAB and Python for Electromagnetic Applications. en. In: *2020 International Applied Computational Electromagnetics Society Symposium (ACES)*. Mar. 2020.

[63] William F. Egan. *Practical RF system design*. New York : Hoboken, N.J: IEEE ; Wiley-Interscience, 2003. 386 pp. ISBN: 978-0-471-20023-9.

[64] Intel. *Intel® Math Kernel Library (Intel® MKL)*. 2020. URL: https://software.intel.com/en-us/mkl (visited on 01/14/2020).

[65] Atef Z. Elsherbeni and Veysel Demir. *The Finite-Difference Time-Domain Method for Electromagnetics with MATLAB® Simulations*. 2 edition. Edison, NJ: Scitech Publishing, Nov. 25, 2015. 560 pp. ISBN: 978-1-61353-175-4.

[66] Alec Weiss, Jeanne Quimby, Rod Leonhardt, Ben Jamroz, Dylan Williams, Kate Remley, Peter Vouras, and Atef Elsherbeni. Configuration and Control of a Millimeter-Wave Synthetic Aperture Measurement System with Uncertainties. In: *2020 95th ARFTG Microwave Measurement Conference (ARFTG)*. 2020 95th ARFTG Microwave Measurement Conference (ARFTG). Los Angeles, CA, USA: IEEE, Aug. 4, 2020. ISBN: 978-1-72810-951-0. DOI: 10.1109/ARFTG47271.2020.9241381. URL: https://ieeexplore.ieee.org/document/9241381/ (visited on 07/13/2021).

[67] Alec Weiss. Characterization of a Vector Network Analyzer Based Millimeter-Wave Channel Sounder. en. MA thesis. Colorado School of Mines, Nov. 2018.

[68] Ruoyu Sun, Peter B. Papazian, Jelena Senic, Yeh Lo, Jae-Kark Choi, Kate A. Remley, and Camillo Gentile. Design and calibration of a double-directional 60 GHz channel sounder for multipath component tracking. In: *2017 11th European Conference on Antennas and Propagation (EUCAP)*. ISSN: null. Mar. 2017, 3336–3340. DOI: 10.23919/EuCAP.2017.7928270.

[69] C. Umit Bas, Rui Wang, Seun Sangodoyin, Dimitris Psychoudakis, Thomas Henige, Robert Monroe, Jeongho Park, Jianzhong Zhang, and Andreas F. Molisch. Real-Time Millimeter-Wave MIMO Channel Sounder for Dynamic Directional Measurements. *arXiv:1807.11921 [cs, eess, math]*, July 2018. arXiv: 1807.11921. URL: http://arxiv.org/abs/1807.11921 (visited on 03/03/2020).

[70] George R. MacCartney and Theodore S. Rappaport. A Flexible Millimeter-Wave Channel Sounder With Absolute Timing. *IEEE Journal on Selected Areas in Communications*, 35(6):1402–1418, June 2017. Conference Name: IEEE Journal on Selected Areas in Communications. ISSN: 1558-0008. DOI: 10.1109/JSAC.2017.2687838.

[71] Sylvain Ranvier, Mikko Kyro, Katsuyuki Haneda, Tuomas Mustonen, Clemens Icheln, and Pertti Vainikainen. VNA-based wideband 60 GHz MIMO channel sounder with 3-D arrays. In: *2009 IEEE Radio and Wireless Symposium*. ISSN: 2164-2974. Jan. 2009, 308–311. DOI: 10.1109/RWS.2009.4957340.

[72] Johannes Hejselbaek, Wei Fan, and Gert F. Pedersen. Ultrawideband VNA based channel sounding system for centimetre and millimetre wave bands. In: *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. ISSN: 2166-9589. Sept. 2016, 1–6. DOI: 10.1109/PIMRC.2016.7794728.

[73] Alec Weiss, Dylan F. Williams, Jeanne Quimby, Rod Leonhardt, Thomas Choi, Zihang Cheng, Kate A. Remley, Andreas Molisch, Benjamin Jamroz, Jake Rezac, Peter Vouras, and Charlie Zhang. Large-Signal Network Analysis for Over-the-Air Test of Up-Converting and Down-Converting Phased Arrays. In: *2019 IEEE MTT-S International Microwave Symposium (IMS)*. ISSN: 0149-645X. June 2019. DOI: 10.1109/MWSYM.2019.8700781.

[74] Dylan F. Williams, J.C.M. Wang, and U. Arz. An optimal vector-network-analyzer calibration algorithm. *IEEE Transactions on Microwave Theory and Techniques*, 51(12):2391–2401, Dec. 2003. Conference Name: IEEE Transactions on Microwave Theory and Techniques. ISSN: 1557-9670. DOI: 10.1109/TMTT.2003.819211.

[75] Jeffrey A. Jargon, Chihyun Cho, Dylan F. Williams, and Paul D. Hale. Physical models for 2.4 mm and 3.5 mm coaxial VNA calibration kits developed within the NIST microwave uncertainty framework. In: *2015 85th Microwave Measurement Conference (ARFTG)*. ISSN: null. May 2015, 1–7. DOI: 10.1109/ARFTG.2015.7162913.

[76] I. Kasa. Closed-Form Mathematical Solutions to Some Network Analyzer Calibration Equations. *IEEE Transactions on Instrumentation and Measurement*, 23(4):399–402, Dec. 1974. Conference Name: IEEE Transactions on Instrumentation and Measurement. ISSN: 1557-9662. DOI: 10.1109/TIM.1974.4314321.

[77] Jan Verspecht. Calibration of a Measurement System for High Frequency Nonlinear Devices, 1995. DOI: 10.1.1.40.1398. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.1398.

[78] Joshua A. Gordon, David R. Novotny, Michael H. Francis, Ronald C. Wittmann, Miranda L. Butler, Alexandra E. Curtin, and Jeffery R. Guerrieri. Millimeter-Wave Near-Field Measurements Using Coordinated Robotics. *IEEE Transactions on Antennas and Propagation*, 63(12):5351–5362, Dec.

2015. Conference Name: IEEE Transactions on Antennas and Propagation. ISSN: 1558-2221. DOI: 10.1109/TAP.2015.2496110.

[79] Dylan F. Williams. *Microwave Uncertainty Framework*. Last Modified: 2019-12-23T17:13-05:00 Library Catalog: www.nist.gov. Aug. 2009. URL: https://www.nist.gov/services-resources/software/wafer-calibration-software (visited on 03/03/2020).

[80] Ulrich Jakobus, Renier G. Marchand, and Daniël J. Ludick. Aspects of and Insights Into the Rigorous Validation, Verification, and Testing Processes for a Commercial Electromagnetic Field Solver Package. *IEEE Transactions on Electromagnetic Compatibility*, 56(4):759–770, Aug. 2014. Conference Name: IEEE Transactions on Electromagnetic Compatibility. ISSN: 1558-187X. DOI: 10.1109/TEMC.2014.2299408.

[81*] Alec Weiss, Atef Elsherbeni, and Jeanne Quimby. Verification of an Evaluator for a New-Radio Channel Estimator. In: *2021 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*. Jan. 2021, 64–65. DOI: 10.23919/USNC-URSINRSM51531.2021.9336510.

[82] Antonius J. van den Biggelaar, Ulf Johannsen, Paul Mattheijssen, and Adrianus B. Smolders. Improved Statistical Model on the Effect of Random Errors in the Phase and Amplitude of Element Excitations on the Array Radiation Pattern. *IEEE Transactions on Antennas and Propagation*, 66(5):2309–2317, May 2018. Conference Name: IEEE Transactions on Antennas and Propagation. ISSN: 1558-2221. DOI: 10.1109/TAP.2018.2800519.

[83] *SciPy.org — SciPy.org*. 2020. URL: https://www.scipy.org/ (visited on 01/22/2020).

[84] *Handbook of mathematical functions: with formulas, graphs, and mathematical tables*. 9th ed. Dover books on mathematics. Dover Publ, 1972. ISBN: 978-0-486-61272-0.

[85] Sympy Development Team. *SymPy*. 2020. URL: https://www.sympy.org/en/index.html.

[86] Peter Lepage. *vegas*. 2021. URL: https://github.com/gplepage/vegas.

[87] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, Mar. 1951. ISSN: 0003-4851. DOI: 10.1214/aoms/1177729694. URL: http://projecteuclid.org/euclid.aoms/1177729694 (visited on 02/23/2021).

[88] A.J. (Teun) van den Biggelaar. *Over-the-air characterization of millimeter-wave integrated antenna systems*. Technische Universiteit Eindhoven, 2020. ISBN: 978-90-386-5125-5.

[89] *NumPy — NumPy*. 2020. URL: https://numpy.org/ (visited on 01/22/2020).

[90] *Numba: A High Performance Python Compiler*. 2020. URL: http://numba.pydata.org/ (visited on 01/22/2020).

[91] NVidia Corporation. *CUDA Zone*. NVIDIA Developer. July 18, 2017. URL: https://developer.nvidia.com/cuda-zone (visited on 01/16/2020).

[92] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations:7, 2020.

[93] Pooria Pakrooh, Arash Amini, and Farokh Marvasti. OFDM pilot allocation for sparse channel estimation. *EURASIP Journal on Advances in Signal Processing*, 2012(1):59, Dec. 2012. ISSN: 1687-6180. DOI: 10.1186/1687-6180-2012-59. URL: https://asp-eurasipjournals.springeropen.com/articles/10.1186/1687-6180-2012-59 (visited on 01/08/2020).

[94] Pramod Mathecken, Taneli Riihonen, Stefan Werner, and Risto Wichman. Constrained Phase Noise Estimation in OFDM Using Scattered Pilots Without Decision Feedback. *IEEE Transactions on Signal Processing*, 65(9):2348–2362, May 1, 2017. ISSN: 1053-587X, 1941-0476. DOI: 10.1109/TSP.2017.2655481. arXiv: 1606.00682. URL: http://arxiv.org/abs/1606.00682 (visited on 12/14/2019).

[95] 3GPP. *5G; NR; Physical Channels and Modulation*. English. Technical Specification 38.211. ETSI, July 2018, 98. URL: https://www.etsi.org/deliver/etsi_ts/138200_138299/138211/15.02.00_60/ts_138211v150200p.pdf (visited on 12/10/2019).

[96] 3GPP. *5G; NR; User Equipment (UE) radio transmission and reception; Part 1: Range 1 Standalone*. Technical Specification. July 2020. URL: https://www.etsi.org/deliver/etsi_ts/138100_138199/13810101/16.04.00_60/ts_13810101v160400p.pdf (visited on 06/01/2021).

[97] FranÇois Septier, Yves Delignon, Atika Menhaj-Rivenq, and Christelle Garnier. Monte Carlo Methods for Channel, Phase Noise, and Frequency Offset Estimation With Unknown Noise Variances in OFDM Systems. en. *IEEE Transactions on Signal Processing*, 56(8):3613–3626, Aug. 2008. ISSN: 1053-587X. DOI: 10.1109/TSP.2008.919629. URL: http://ieeexplore.ieee.org/document/4567677/ (visited on 02/25/2020).

[98] Hong Wan, Rong-Rong Chen, Jun Won Choi, Andrew Singer, James Preisig, and Behrouz Farhang-Boroujeny. Joint channel estimation and Markov Chain Monte Carlo detection for frequency-selective channels. en. In: *2010 IEEE Sensor Array and Multichannel Signal Processing Workshop*. Jerusalem, Israel: IEEE, Oct. 2010, 5606768. ISBN: 978-1-4244-8978-7. DOI: 10.1109/SAM.2010.5606768. URL: http://ieeexplore.ieee.org/document/5606768/ (visited on 07/30/2021).

[99] Alec Weiss, Atef Elsherbeni, Jeanne Quimby, and Jacob Rezac. Framework for Evaluating Channel Estimators in Millimeter-Wave New-Radio Systems. en. *Accepted for Presentation at the 2022 IEE International Symbosium on Antennas and Propagation July, 2022.*

[100*] Alec Weiss, Atef Elsherbeni, Jeanne Quimby, and Jacob Rezac. Monte Carlo Augmented Channel Estimator. en. *Accepted for publication in URSI Radio Science Letters on April 11, 2022.*

[101] Keysight Technologies. *IQ Gain Imbalance and Quadrature Skew Error Data Interaction*. 2021. URL: https://rfmw.em.keysight.com/wireless/helpfiles/89600b/webhelp/subsystems/digdemod/Content/digdemod_para_interact_iqgainimb_quadskewerr.htm.

[102] Tuan Anh Le, Atilim Gunes Baydin, Robert Zinkov, and Frank Wood. Using Synthetic Data to Train Neural Networks is Model-Based Reasoning. *arXiv:1703.00868 [cs, stat]*, 2017. arXiv: 1703.00868. URL: http://arxiv.org/abs/1703.00868.

APPENDIX A

COORDINATE SYSTEMS

The coordinate systems in this paper follow the conventions defined in this section. While there are many possible ways to define these coordinate systems these were chosen due to their ease of use with how phased arrays have been defined. Phased arrays in this dissertation are defined with the z axis as boresight. For example a planar array would typically be defined by elements on the xy plane. It is possible for either of these coordinate systems to be converted to or from one another and therefore typically the most intuitive coordinate system is used for a problem and converted if needed.

A.1 Coordinates (θ, ϕ)

Spherical coordinate systems referenced by (θ, ϕ) are defined with polar angle θ defined from (0,π) and azimuthal angle ϕ defined from (0,2π). This coordinate system has the θ value coming down from the z axis. The ϕ value swings around the z axis in the xy plane, starting from the positive x axis. A visualization of this coordinate system is seen in Figure A.1.

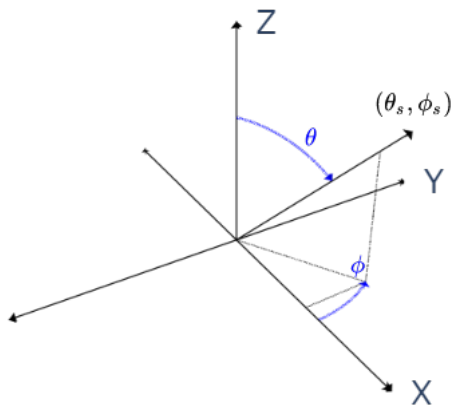


Figure A.1 Visualization of coordinate system defined by (θ, ϕ).

APPENDIX B

CODE LISTINGS AND BASIC USAGE

B.1 Hardware Component Model

Each hardware component (both parametrically defined and S-parameter defined) derive from a base class `HardwareComponent`. This class is derived from a dictionary to store input and state information on the device as it is simulated. The code for the base class definition is seen in Listing B.1.

Listing B.1: Base hardware component definition.

```
B.1  '''
B.2  class HardwareComponent(dict):
B.3      '''
B.4      @brief class to hold a model (virtual representation) of a hardware component (
             physical device)
B.5      @note funct removed. Now override self._function
B.6      @param[in] parameters - dict with {'{name}':{dtype},...} specifying component
             parameters
B.7      @param[in/OPT] shape - number of repeat components to make in this class. e.g.
             size of allocated arrays.
B.8          If not provided, set to none and wait until self.set_shape() has been called
B.9      @param[in/OPT] noise_flg - whether or not this component will have noise (
             default false)
B.10     @note The returned object will be callable
B.11     '''
B.12     def __init__(self,name:str='',parameters:dict={},shape:int=None,noise_flg:bool=
             False,
B.13                  freqs=None,verbose=False,engine='py',**kwargs):
B.14         '''@brief Constructor'''
B.15         super().__init__(**kwargs)
B.16         self['id'] = name
B.17         self['name'] = name # keep for backwards compatability
B.18         self['parameters'] = parameters
B.19         self['noise_flg'] = noise_flg
B.20         self['freqs'] = freqs
B.21         self['verbose'] = verbose
B.22         self['engine'] = None # initial state
B.23         self.add_noise_override=None
B.24         # check if we want to override our noise function before setting our engine
B.25         if callable(noise_flg):
B.26             self.add_noise_override = noise_flg
B.27             self['noise_flg'] = True
B.28         # Set shape (and allocate if not None)
B.29         self.set_shape(shape)
B.30         self.set_engine(engine)
B.31
B.32         # frequency mask for masked calculations (only some frequencies)
B.33         self.set_freq_mask() # reset masking
```

This class is supported by many methods to perform various operations from slicing of arrays to setting different computational engines. Arguably, the most important methods are the `_add_noise_<engine>`,

`_frequency_response_<engine>`, and `_function_<engine>`. These define the function to add noise to the components response, the frequency response of the component, and a function of how the input of the component maps to the output for a given `<engine` (e.g. `py` for python). These methods are defined in the base class in Listing B.2

Listing B.2: Base hardware component important default methods.

```
B.1         @note this will add a the 'parameters' key to the self
B.2         @return self with allocated arrays in the 'parameters' key
B.3         '''
B.4         eng_lib = np if self['engine']!='gpu' else cp
B.5         # Allocate the parameter arrays if not initialized
B.6         for k,v in self['parameters'].items():
B.7             if isinstance(v,type): # then assume it hasnt been allocated
B.8                 self['parameters'][k] = eng_lib.ndarray(self['shape'],dtype=v)
B.9             elif reallocate: #otherwise assume it exists but we want to reallocate (
                    with current dtype)
B.10                self['parameters'][k] = eng_lib.ndarray(self['shape'],dtype=self['
                        parameters'][k].dtype)
B.11
B.12    def _add_noise_py(self,data):
```

B.1.1 Parametric (Equation) Based Hardware Component Model

Some hardware component models are defined by equations. An example of this is given by the "generic" continuous phase shifter definition listed in Listing B.3. We can see by this that the base methods are overridden to provide noise and frequency responses specific to the phase shifter. For example, the `_frequency_response_<engine>` method has been overriden to provide a phase shift based on the parameters set for the component.

Listing B.3: Definition and methods of a generic phase shifter.

```
B.1  class PhaseShifter(HardwareComponent):
B.2      '''
B.3      @brief This is a Component for a generic phase shifter with ideal shifts
B.4      @param[in] args - arguments passed to super().__init__
B.5      @param[in] kwargs - values override defaults and passed to super().__init__
B.6      @note this can be inherited from as a base level phase shifter
B.7      '''
B.8      component_info = {'name':'phase_shifter_ideal','parameters':{'shift':np.cdouble
             }}
B.9
B.10     def __init__(self,*args,noise_flg=False,**kwargs):
B.11         '''@brief constructor'''
B.12         component_kwargs = copy.deepcopy(self.component_info)
B.13         component_kwargs.update(kwargs)
B.14         super().__init__(noise_flg=noise_flg,**component_kwargs)
B.15
B.16     def _add_noise_py(self,data):
B.17         '''
```

```
B.18            @brief Add noise from phase shifter.
B.19            @note only phase noise is added to separate phase/magnitude
B.20            @param[in] data - data to add noise onto
B.21            '''
B.22            return add_phase_noise(data,lambda shape: np.random.normal(0,np.deg2rad(5),
                    shape))
B.23
B.24        def set_shift(self,shift):
B.25            '''@brief set the shifts for the component. Can be scalar, or list with
                    shape self['shape']'''
B.26            self.set_parameter('shift', shift)
B.27
B.28        def _frequency_response_py(self):
B.29            '''@brief get the frequency response'''
B.30            return np.exp(1j*self['parameters']['shift'][...,np.newaxis])
B.31
B.32        def _frequency_response_gpu(self):
B.33            return cp.exp(1j*cp.asarray(self['parameters']['shift'])[...,cp.newaxis])
```

This is then taken one step further to provide discretization of the phase shifter. Listing B.4 provides the definition and method overrides to simulate a discretized phase shifter of a given number of bits. This is achieved by rounding to the nearest angle after dividing 2π by the number of possible phase shifter states.

Listing B.4: Definition and methods of a discretized phase shifter.

```
B.1         '''
B.2         @brief Class to mimic a discretized phase shifter with nbits
B.3         @param[in] nbits - number of bits in the phase shifter
B.4         @param[in] args - arguments passed to super().__init__
B.5         @param[in/OPT] kwargs - kwargs passed to super.__init__
B.6         '''
B.7         def __init__(self,nbits,noise_flg=False,*args,**kwargs):
B.8             '''@brief constructor'''
B.9             super().__init__(*args,
B.10                             name='phase_shifter_{}bits'.format(nbits),**kwargs)
B.11            self['nbits'] = nbits
B.12            self['noise_flg'] = noise_flg
B.13
B.14        def _frequency_response_py(self):
B.15            '''@brief get the frequency response with nbits'''
B.16            nstates = 2**self['nbits']
B.17            round_val = 2*np.pi/nstates
B.18            # now lets calculate our phases and round
B.19            phases = np.exp(1j*self['parameters']['shift'])
B.20            return round_angle(phases,round_val)[...,np.newaxis]
B.21
B.22        def _frequency_response_gpu(self):
B.23            '''@brief get the frequency response with nbits'''
B.24            nstates = 2**self['nbits']
B.25            round_val = 2*np.pi/nstates
B.26            # now lets calculate our phases and round
B.27            phases = cp.exp(1j*self['parameters']['shift'])
B.28            return round_angle(phases,round_val,engine=cp)[...,cp.newaxis]
```

These equation based components provide the ability to easily define and simulate hardware components without the need for direct measurements.

B.1.2 S-parameter (Measured) based Hardware Component Model

Hardware component models based on measured S-parameter components were derived from a base class that handled necessary operations such as loading, parsing, and organizing measured data. The definition of this class is given by Listing B.5.

Listing B.5: Base S-parameter (measured) hardware component definition.

```
B.1  class SParameterComponent(HardwareComponent):
B.2      '''
B.3      @brief class to hold a hardware component from 2 port S-parameters (must be
             cascaded)
B.4      @param[in] snp - SnpEditor object to create the component from
B.5      @parma[in/OPT] freqs - frequencies to interpolate to. If none, just load the
             data
B.6      @note The returned object will be callable
B.7      '''
B.8      def __init__(self,snp:SnpEditor,freqs=None,**kwargs):
B.9          '''@brief Constructor'''
B.10         # load in the s parameter data
B.11         if not isinstance(snp,TouchstoneEditor): snp = SnpEditor(snp) #if its a
                 string assume its a path
B.12         freq_fun = lambda x:x
B.13         if freqs is not None: freq_fun = lambda x: interpolate(x,freqs)
B.14         # now lets set our class values
B.15         self['touchstone'] = freq_fun(snp) # the raw touchstone parameter
B.16         # Now initialize parent. We need to set self['touchstone'] before running
                 self.set_shape() or self._set_data
B.17         super().__init__(**kwargs)
B.18         # values for noise
B.19         default_noise = {11:None,21:None,12:None,22:None}
B.20         if hasattr(self,'_add_noise'): default_noise[21] = self._add_noise; Logger.
                 debug('Using _add_noise on S21 for {}'.format(self.get('name')))
B.21         else: Logger.debug("No _add_noise defined for {}".format(self.get('name')))
B.22         self.noise_functs = {11:None,21:getattr(self,'_add_noise'),12:None,22:None}
                 # default s21 to self._add_noise(_py/_gpu)
B.23         # set up variables for data
B.24         self.data = None # all of the data for the array
```

For devices like phase shifters, an extension of this class was made to allow for multiple states. This allows for loading multiple measurements from different device states and easily changing between measured device states. The code for this class is seen in Listing B.6

Listing B.6: Multi-state S-parameter (measured) hardware component definition.

```
B.1
B.2  class MultistateSParameterComponent(SParameterComponent):
B.3      '''
```

```
B.4       @brief Component for easily implementing SParameter components with multiple
              states (e.g. phase shifers/amplifiers)
B.5       @param[in] snp_list - list of SnpEditors or path names (strings) to load into
              our states
B.6       @note This sets self['touchstone'] to snp_list[0] to allow reuse of
              SParameterComponent methods without change
B.7       '''
B.8       def __init__(self,snp_list:list,freqs=None,**kwargs):
B.9           '''@brief constructor'''
B.10          # Makes ure we have SnpEditor objects
B.11          for i,snp in enumerate(snp_list):
B.12              if not isinstance(snp,TouchstoneEditor):
B.13                  snp_list[i] = SnpEditor(snp) #assume its a string or something
B.14          # Get desired frequencies
B.15          freq_fun = lambda x:x
B.16          if freqs is not None: freq_fun = lambda x: interpolate(x,freqs)
B.17          snp_list = [freq_fun(snp) for snp in snp_list]
B.18          super().__init__(snp_list[0],**kwargs) # init parent
B.19          self.states = snp_list
B.20
B.21      def set_states(self,state_idx=None):
B.22          '''@brief set self.data from the states for a list state_idx, if None, use
                  self._get_state_idx()'''
B.23          if state_idx is None: state_idx = self._get_state_idx()
B.24          if np.ndim(state_idx)==0: state_idx = [state_idx]
B.25          reshaped_data = self.data.reshape((-1,*self.data.shape[-2:])) # collapse
                  array shape to 1D
B.26          snp_template = SnpEditor((self['touchstone'].num_ports,self['touchstone'].
                  freqs)) # template format
B.27          snp_list = []
B.28          for i in range(len(reshaped_data)): #each of these represents a component of
                   a single element
B.29              reshaped_data[i,:,:] = self.states[state_idx[i]].to_numpy()
B.30          self._get_touchstone() # now get the touchstone object of each element
B.31
B.32      def _set_data(self,shape):
B.33          '''@brief also reset the states when the data is set to a new shape from
                  super'''
B.34          super()._set_data(shape)
B.35          self.set_states()
```

B.1.3 Grouping Hardware Component Models

After the models themselves are defined, they can be grouped to be run in a serial fashion. This makes it easy to quickly define and simulate things like phased arrays. The base class for this definition is given as `HardwareComponentGroup` in Listing B.7. These groupings simulate the devices and link them together serially so the output of each component is automatically fed into the input of the next. This is also used to compile functions from these devices together to increase computation speed. These groups can be quickly generated by providing a list of `HardwareComponent` classes. For example, we could generate a phased array to simulate with:

```
B.1   myarray = HardwareComponentGroup([Amplifier(),PhaseShifter(),Combiner()])
```

Listing B.7: Definition of hardware component groups for serial simulation.

```
            obj_list.append(cascade)
        extra_data['objects'] = obj_list

        # now get the functions
        funct_list = [obj._function for obj in obj_list]
        inv_funct_list = [obj._function_inv for obj in obj_list[::-1]]

        # now lets wrap our functions for debuggina and whatnot
        funct_list_wrapped = []
        for i,f in enumerate(funct_list): # wrap our functions
            if int(verbose)>0: # verbosity 2 or greater
                f=debug_wrap(f,'Simulating {}'.format(obj_list[i].get('name','NA')))
            funct_list_wrapped.append(f)

        def funct(*data): # cascade the data
            return functools.reduce(lambda d,fun: fun(*d) if isinstance(d,tuple) else
                fun(d),funct_list_wrapped,data)
        # inverse simulation
        def funct_inv(*data): # cacade in reverse
            return functools.reduce(lambda d,fun: fun(*d) if isinstance(d,tuple) else
                fun(d),inv_funct_list,data)

        # and return our values
        return funct,funct_inv,extra_data

    #function for wrapping debug
    def debug_wrap(fun,str):
        def wrapper(*args,**kwargs):
            Logger.debug(str)
            return fun(*args,**kwargs)
        return wrapper

    #%% Class to hold multiple hardware components
    '''
    AS OF 2/18/2021 ---
    Connections will be defined by the following:
        'src_component_id':{'id':'dst_component_id'}

    This will allow future expansion to multiport/etc by using a dict.
            right now this is a list of components to run in order (e.g.)
        @note This will also store an internal mapping of each component (subcomponent)
        '''
        def __init__(self,components,*args,connections=None,parallel=False,engine='py',
            verbose=False,**kwargs):
```

.

B.2 Channel Estimators

Like the hardware models, each defined channel estimator inherited from a base class. In the case of channel estimators, this was simply to standardize the interfaces (i.e. input parameters and output values) of each method. This base definition is required to operate correctly with the channel estimation evaluation

framework in this dissertation. The definition of the base class is seen in Listing B.8.

Listing B.8: Defnition of the base class for channel estimators.

```
class ChannelEstimator(dict):
    '''
    @brief base class for a channel estimator
    @note Instantiations of this class must be callable for the estimator
    @note this inherits from a dictionary to easily store configurations directly in the class
    @param[in/OPT] symbol_slice - default symbols to slice data from x and y variables during call
    '''
    def __init__(self,*args,symbol_slice:slice=slice(None,None,None),**kwargs):
        '''@brief constructor'''
        if 'name' not in kwargs.keys(): kwargs['name'] = 'NAME NOT DEFINED'
        super().__init__(*args,**kwargs)
        self._estimate=None
        self['symbol_slice'] = symbol_slice

    def __call__(self,x:typing.Iterable[complex],y:typing.Iterable[complex],symbol_slice:slice=None) -> typing.Iterable[complex]:
        '''
        @brief function run when calling the instance of the class
        @param[in] x - transmitted (ideal) sent pilot values for a single ofdm symbol
            from pilot info ({'idx':(iidx,jidx),'value':values}) iidx=symbol #, jids=subcarrier #
        @param[in] y - recieved values at each pilot tone for one OFDM symbol
            from pilot info ({'idx':(iidx,jidx),'value':values})
        @param[in] symbol_slice - what symbols to calculate estimate from.
        @return estimated complex channel response dictionary (allows extra values)
            {'idx':(iidx,jidx),'value':response,...any other data}
        '''
        # Calculate the estimate (if its not already there)
```

An example of using this base class can be seen by the simple least squares estimator. The definition of a least squares estimator can be seen in Listing B.9.

Listing B.9: Defnition of a basic least squares (LS) channel estimator.

```
        xslice = self.get_symbol_slice(x, symbol_slice)
        yslice = self.get_symbol_slice(y, symbol_slice)
        self._estimate = self.estimate(xslice,yslice)
        # return our estimate. This will be expanded in interpolation
        return self._estimate

    def reset(self):
        '''@brief do some things to reset. If not implemented do nothing'''
        self._estimate=None

    def set_defaults(self,**kwargs):
        '''@brief set default values'''
        for k,v in kwargs.items():
            self[k] = v

```

```
B.16        def estimate(self,x:typing.Iterable[complex],y:typing.Iterable[complex]) ->
                typing.Iterable[complex]:
B.17            '''@brief the function to actually perform the estimation. This will perform
                     the estimate at self._estimate'''
B.18            raise NotImplementedError
B.19
B.20        @staticmethod
B.21        def get_symbol_slice(pilot_info:dict,myslice:slice):
B.22            '''@brief given a slice of symbols, extract the pilot info from a {'idx':(
                    iidx,jidx),'value':values} type layout'''
B.23            # first get our number of symbols
B.24            max_sym = int(np.max(pilot_info['idx'][0])+1)
B.25            min_sym = int(np.min(pilot_info['idx'][0]))
B.26            sym_nums = list(range(min_sym,max_sym))[myslice]
B.27            # now extract the indices for each of these
B.28            info_idx = np.logical_or.reduce([pilot_info['idx'][0]==v for v in sym_nums])
B.29            return {'idx':(pilot_info['idx'][0][info_idx],pilot_info['idx'][1][info_idx
                    ]),
B.30                    'value':pilot_info['value'][info_idx]}
B.31
B.32
B.33    #%% Least squares estimator (just y/x)
B.34    class LS(ChannelEstimator):
B.35        '''
B.36        @brief Class for a least squares estimator (LS). Can be called once instantiated
B.37        @param[in/OPT] nsamples - How many samples to use (take mean of for each
                subcarrier). 'all' is acceptable input
B.38        @note This is calculated as y=Xh so h[i] = rx[i]/tx[i]
B.39        '''
B.40        def __init__(self,*args,nsamples=1,**kwargs):
B.41            '''@brief constructor'''
```

B.2.1 Monte Carlo Augmented Channel Estimator

The MCCE class contains many extra methods to perform things like the calculation of the unmodeled and total responses. The definition for this class is shown in Listing B.10. This definition also includes a static variable to define default configuration values of the MCCE.

Listing B.10: Definition of the Monte Carlo augmented channel estimator class.

```
B.1     class MCCE_LS(ChannelEstimator):
B.2         '''
B.3         @brief Correct data with a monte carlo based channel estimator (MCCE)
B.4         @param[in] array - AnalogAntennaArray object. This will be used for monte-carlo
                simulations
B.5             so will typically change on each call to .simulate()
B.6         @param[in/OPT] verbose - whether to be verbose (default False)
B.7         @param[in/OPT] config - overriding of default configuration
B.8         @return Mean of monte carlo channel estimates h_mc
B.9         '''
B.10
B.11        DEFAULT_CONFIG = {
B.12            # Input parameters by the user
B.13            'array':None,
B.14            # monte carlo iterations for forward propagations
```

```
B.15            'mc_iterations':1000,
B.16            'base_estimator':LS(nsamples='all'),
B.17            # response of any unmodeled (Hota for now). abbreviated umr
B.18            ## The function can be overriden as a callable() to return the unmodeled (e.
                    g. hota) response
B.19            ## if function is None, one will be populated using _get_unmodeled_ls with a
                     provided # of iterations
B.20            'unmodeled_response': {'function':None,'iterations':20000},
B.21            # whether or not to be verbose
B.22            'verbose':True ,
B.23            'chunk_size':100,
B.24            }
B.25
B.26        def __init__(self,array:AnalogAntennaArray,symbol_slice:slice=slice(None,None,
                None),verbose:bool=True,**config):
B.27            '''@brief constructor'''
B.28            super().__init__(name='Monte-Carlo Channel Estimator (MCCE)',symbol_slice=
                    symbol_slice)
B.29            self.update(copy.deepcopy(self.DEFAULT_CONFIG))
B.30            self['verbose'] = verbose
B.31            # other configs
B.32            for k,v in config.items():
B.33                self[k] = v
B.34            # now set the array for the system
B.35            self.set_array(array)
B.36
B.37
B.38        def set_array(self,array:AnalogAntennaArray):
B.39            '''@brief set a new array for the estimator'''
B.40            self['array'] = array
B.41            ## This will be called with .simulate(1) repeatedly to get noise
                    distributions from models
B.42            self._array_unmodeled = None
B.43            ## we need the array without an antenna for simulation (channel estimate
                    will include antenna)
```

.

Another interesting method that is unique to the MCCE code is the calculation of the unmodeled response (e.g., UMR, $\hat{\mathbf{h}}_{umod}$). The code used to calculate this response can be seen in Listing B.11

Listing B.11: Method to calculate the unmodeled response of the Monte Carlo augmented channel estimator.

```
B.1             conv = get_convergence(resp_vals,mc_est_fun=est_fun,verbose=verbose,**kwargs
                    )
B.2             return conv
B.3
B.4         def _get_unmodeled_ls(self,x,y,num_mc:int=None,getfunct:bool=True,base_estimator
                =None,
B.5                               mc_chunk_size=None) -> np.ndarray:
B.6             '''
B.7             @brief get Calculate unmodeled (typically Hota) distribution for each
                    element
B.8             @param[in] x - tx x value info
B.9             @param[in] y - rx y value info {'idx':(iidx,jidx),'value':values}
B.10            @@param[in/OPT] num_mc - number of Monte Carlo simualtions to run for calc
                    of hota
```

```
B.11        @param[in/OPT] getfunct - whether to return a callable (true)
B.12            or a list of the calculated values(false) (default true)
B.13        @param[in/OPT] base_estimator - base estimator to perform monte-carlo off of
                .
B.14            if None, default to self['base_estimator']
B.15        @note This is calculated in a multistep proscess by approximating Hota using
B.16            a least squares approximation with an overdefined system of equations.
                    This
B.17            will allow us to account for the bias from combination
B.18            This will be of the form A@x=b where A and b are known, but A is NOT a
                    full
B.19            rank square matrix (non-invertible)
B.20            1. AT EACH FREQUENCY create a b vector of our resulting output
B.21            2. Create an A matrix by sampling noise PRIOR to our combination
B.22            3. Solve this to get our values of x which should be an approximation of
                    our channel
B.23        @TODO add support for all frequencies (currently hardcoded freq 0)
B.24        @return function that randomly samples the calculated set of hota values
B.25        '''
B.26
B.27        if base_estimator is None: base_estimator = self['base_estimator']
B.28        hbase = base_estimator(x,y) # perform our base estimation
B.29        # Clean our input args
B.30        xvals = np.asarray(x['value']); yvals = np.asarray(y['value'])
B.31        # Some setup
B.32        if num_mc is None: num_mc = self['unmodeled_response']['iterations']
B.33        ## if x is a list of packets and y is a single, assume multiple samples
B.34        if np.ndim(yvals)>1 and np.ndim(yvals)>=np.ndim(xvals):
B.35            num_samps = len(yvals)
B.36        else:
B.37            num_samps = 1
B.38
B.39        umr_fvals = []
B.40        # calculate noise response for every one of our frequencies (simulatneous
                for s-parameter based things)
B.41        resp_vals = self._get_array_unmodeled(num_mc, nfreqs=np.max(x['idx'][-1])+1)
B.42
B.43        ## now iterate through freqs
B.44        fidx = np.unique(x['idx'][-1])
B.45        for fi in fidx: # loop through each indexed frequency (subcarrier)
B.46            # Step 1 - normalize and create our b vectors
B.47            yvalsf = y['value'][y['idx'][-1]==fi] # get the values only where the
                    indices match subcarriers
B.48            # Step 2 - hardcoded for only amp and ps right now. remove combiner
B.49            # MOVED TO BEFORE LOOP
B.50
B.51            ## Now make into our A matrix - Get the error from the uncombined array
B.52            A = resp_vals[...,fi] # get the values for our frequency (subcarrier)
B.53
B.54            # Step 3 - Perform LS approximation
B.55            umr_ls_rv = np.linalg.lstsq(A,b,None)
B.56            umr_ls = umr_ls_rv[0] # ignore the residual for now (just get LS
                    estimate)
B.57            umr_fvals.append(umr_ls)
B.58
B.59        # now repack this will be like (pilot idx, nelems)
B.60        umr_fvals = np.asarray(umr_fvals)#.T # change so umr_fvals[0] is our fresp
                at element 0 (NOT ANYMORE!)
```

```
B.61            umr_rv = {'idx':x['idx'],'value':[umr_fvals[np.where(fidx==xi)[0][0]] for xi
                    in x['idx'][1]]} # value here is list of estimated channel response at
                   each frequency index 'idx'
B.62
B.63            # now return
B.64            if getfunct:
```

.

Finally, we can look at the method that calculates the entire MCCE estimation. This method is defined as `__call__` to allow us to use the estimator as `MCCE(x,y)` as opposed to having to fully call a method of the class. The definition of this method is seen in Listing B.12

Listing B.12: Method to calculate a channel estimate using the Monte Carlo augmented channel estimator.

```
B.1             y_agg['idx'] = y_mc['idx']
B.2             return y_agg
B.3
B.4         def estimate(self,x,y,num_mc:int=None,umr:callable=None):
B.5             '''
B.6             @brief calculate the estimate of the channel
B.7             @param[in] x - input values x
B.8             @param[in] y - corresponding output values y
B.9             @param[in/OPT] num_mc - number of monte_carlo runs to use
B.10            @param[in/OPT] umr - override unmodeled response function
B.11            @return estimate of full transmit (x) to recieve (y) channel estimate
B.12                including uncertainties and bias from hardware
B.13            @note inputs should be {'idx':(symbol_#,subcarrier_#),'value':values}
B.14            '''
B.15            if num_mc is None: num_mc = self['mc_iterations']
B.16            if umr is None: # do this here so we only calculate umr once
B.17                umr = self.get_umr(x,y)
B.18                self['umr'] = umr # save for debug
B.19            y_mc = self.get_monte_carlo_outputs(x,num_mc=num_mc,umr=umr)
```

.

PUBLICATIONS

These are publications that have been presented at conferences or submitted to journals during my academic career. While each of the publications provided a knowledge base for the work in this dissertation, those directly related to this work are denoted with an *****.

Journals

[1] Ryan Smith, **Alec Weiss**, Ravi Bollimuntha, Sanjay DMello, Melinda Piket-May, Mohammed Hadi, and Atef Elsherbeni. Merging VSim's Model Building and Visualization Tools with Custom FDTD Engines. *Applied Computational Electromagnetics Society Journal*, 32(12):1144–1147, 2017.

[2] **Alec Weiss**, Atef Z Elsherbeni, Veysel Demir, and Mohammed F Hadi. Using MATLAB's Parallel Processing Toolbox for Multi-CPU and Multi-GPU Accelerated FDTD Simulations. en. *Advanced Computational Electromagnetics Society Journal*, 34(5):724–730, May 2019.

[3] **Alec Weiss** and Atef Z Elsherbeni. Performance of MATLAB and Python for Computational Electromagnetic Problems. en. *Advanced Computational Electromagnetics Society Journal*, 35(7):770–777, 2020.

[4] Nina B Popovic, Evan A Schlomann, **Alec Weiss**, Ross A Rentz, Edward J Garboczi, Nathan D Orloff, and Christian J Long. Microwave Measurements for Conductive Anisotropic Materials. *IEEE Transactions on Microwave Theory and Techniques*, 68(11):4913–4924, 2020.

[5*] **Alec Weiss**, Atef Elsherbeni, Jeanne Quimby, and Jacob Rezac. Monte Carlo Augmented Channel Estimator. en. *Accepted for publication in URSI Radio Science Letters on April 11, 2022.*

Conferences

[1] **Alec Weiss**, Mohammed Hadi, and Atef Elsherbeni. Performance Analysis of Parallel Higher Order FDTD Methods. en. In: *2018 Graduate Research and Discovery Symposium (GRADS)* (Colorado School of Mines, Golden, Colorado). Colorado School of Mines, Apr. 2018.

[2] Peter Vouras, **Alec Weiss**, Maria Becker, Ben Jamroz, Jeanne Quimby, Dylan Williams, and Kate Remley. Gradient-Based Solution of Maximum Likelihood Angle Estimation For Virtual Array Measurements. In: *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE. 2018.

[3] **Alec Weiss**, Atef Elserbeni, Veysel Demir, and Mohammed Hadi. Accelerating the FDTD Algorithm on CPUs with MATLAB's Parallel Computing Toolbox. In: *2019 International Applied Computational Electromagnetics Society Symposium (ACES)*. Apr. 2019.

[4] **Alec Weiss**, Dylan F. Williams, Jeanne Quimby, Rod Leonhardt, Thomas Choi, Zihang Cheng, Kate A. Remley, Andreas Molisch, Benjamin Jamroz, Jake Rezac, Peter Vouras, and Charlie Zhang. Large-Signal Network Analysis for Over-the-Air Test of Up-Converting and Down-Converting Phased Arrays. In: *2019 IEEE MTT-S International Microwave Symposium (IMS)*. ISSN: 0149-645X. June 2019. DOI: 10.1109/MWSYM.2019.8700781.

[5] Jeanne Quimby, David G Michelson, Mustapha Bennai, Kate Remley, Joshua Kast, and **Alec Weiss**. Interlaboratory millimeter-wave channel sounder verification. In: *2019 13th European Conference on Antennas and Propagation (EuCAP)*. IEEE. 2019.

[6] **Alec Weiss** and Atef Elsherbeni. Computational Performance of MATLAB and Python for Electromagnetic Applications. en. In: *2020 International Applied Computational Electromagnetics Society Symposium (ACES)*. Mar. 2020.

[7] **Alec Weiss** and Atef Elsherbeni. Comparing Runtimes of Python and MATLAB for Computational Electromagnetic Problems. In: *2020 Graduate Research and Discovery Symposium (GRADS)* (Colorado School of Mines, Golden, Colorado). Colorado School of Mines, Apr. 2020. URL: https://youtu.be/0XeGVWLHnGE.

[8] **Alec Weiss**, Jeanne Quimby, Rodney Leonhardt, Robert Jones, Josh Kast, Benjamin Jamroz, Peter Vouras, Dylan Williams, and Kate A. Remley. Millimeter-Wave High Multipath Channel Measurements. en. In: *2020 XXXIIIrd General Assembly and Scientific Symposium of the International Union of Radio Science*. Rome, Italy: IEEE, Aug. 2020. ISBN: 978-94-6396-800-3. DOI: 10.23919/URSIGASS49373.2020.9232158. URL: https://ieeexplore.ieee.org/document/9232158/ (visited on 03/24/2021).

[9*] **Alec Weiss**, Atef Elsherbeni, and Jeanne Quimby. Verification of an Evaluator for a New-Radio Channel Estimator. In: *2021 United States National Committee of URSI National Radio Science Meeting (USNC-URSI NRSM)*. Jan. 2021, 64–65. DOI: 10.23919/USNC-URSINRSM51531.2021.9336510.

[10*] **Alec Weiss** and Atef Elsherbeni. Accelerating Simulations of Large Scale Phased- Array Systems. In: *2021 International Applied Computational Electromagnetics Society Symposium (ACES)*. 2021. URL: https://aces-society.org/conference/Online-Live_2021/.

[11] **Alec Weiss**, Atef Elsherbeni, Jeanne Quimby, and Jacob Rezac. Framework for Evaluating Channel Estimators in Millimeter-Wave New-Radio Systems. en. *Accepted for Presentation at the 2022 IEE International Symbosium on Antennas and Propagation July, 2022.*

Worshops

[1] Akbar Sayeed, Peter Vouras, Camillo Gentile, **Alec Weiss**, Jeanne Quimby, Zihang Cheng, Bassel Modad, Yuning Zhang, Chethan Anjinappa, Fatih Erden, et al. A Framework for Developing Algorithms for Estimating Propagation Parameters from Measurements. In: *2020 IEEE Globecom Workshops (GC Wkshps)*. IEEE. 2020.

[2] Peter Vouras, Jeanne Quimby, Benjamin Jamroz, **Alec Weiss**, Rodney Leonhardt, Dylan F Williams, and Kate A Remley. Frequency Invariant Beampatterns for Wideband Synthetic Aperture Channel Sounders. In: *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE. 2020.

Book Chapters

[1] Peter Vouras, Damla Guven, Jack Chuang, Camillo Gentile, Kate A. Remley, Benjamin Jamroz, Dylan F. Williams, Jeanne Quimby, Rodney Leonhardt, Robert D. Horansky, Maria Becker, **Alec Weiss**, Robert D. Jones, Joshua Kast, Atef Elsherbeni, and Paritosh Manurkar. Over-the-air testing with synthetic-aperture techniques. en. *Metrology for 5G and Emerging Wireless Technologies*:327–368, Dec. 2021. Publisher: IET Digital Library. DOI: 10.1049/PBTE099E_ch11. URL: https://digital-library.theiet.org/content/books/10.1049/pbte099e_ch11;jsessionid=6n8l766bbh73g.x-iet-live-01 (visited on 04/21/2022).

[2] **Alec Weiss** and Atef Elsherbeni. Acceleration of FDTD Code Using MATLAB's Parallel Computing Toolbox. In: *Advances in Time-Domain Computational Electromagnetic Methods*. submitted for review. Wiley.

Thesis

[1] **Alec Weiss**. Characterization of a Vector Network Analyzer Based Millimeter-Wave Channel Sounder. en. MA thesis. Colorado School of Mines, Nov. 2018.