VIABILITY AND PERFORMANCE OF RF SOURCE LOCALIZATION USING AUTOCORRELATION-BASED FINGERPRINTING

by

Joseph L. Ipson

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

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

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

UTAH STATE UNIVERSITY
Logan, Utah

2023

ABSTRACT

Viability and performance of RF source localization using autocorrelation-based fingerprinting

by

Joseph L. Ipson, Master of Science

Utah State University, 2023

Major Professor: Todd K. Moon
Department: Electrical and Computer Engineering

Identifying the location of a radio-frequency (RF) transmitter is a useful capability for many civilian, industrial, and military applications. This problem is made more challenging when done "Blind," or when the transmitter is noncooperative, and relatively little information is available about the signal beforehand. Typical methods for this operation utilize the time, phase, power, and frequency viewable from received signals. These features all behave less predictably in indoor and urban environments, where signals undergo transformation from multiple interactions with the environment. These interactions imprint structure onto the received signal which is dependent on the transmission path, and therefore the initial location. Using a received signal, a signal autocorrelation can be computed which will largely be shaped by this information. In this research, RF source localization using fingerprinting with signal autocorrelations is explored. A Gaussian-process-based method for autocorrelation-based fingerprinting is proposed. Performance of this method is evaluated using a ray-tracing-based simulation of an indoor environment.

(183 pages)

PUBLIC ABSTRACT

Viability and performance of RF source localization using autocorrelation-based fingerprinting

Joseph L. Ipson

Finding the source location of a radio-frequency (RF) transmission is a useful capability for many civilian, industrial, and military applications. This problem is particularly challenging when done "Blind," or when the transmitter was not designed with finding its location in mind, and relatively little information is available about the signal beforehand. Typical methods for this operation utilize the time, phase, power, and frequency viewable from received signals. These features are all less predictable in indoor and urban environments, where signals undergo transformation from multiple interactions with the environment. These interactions imprint structure onto the received signal which is dependent on the transmission path, and therefore the initial location. Using a received signal, a signal characteristic known as the autocorrelation can be computed which will largely be shaped by this information. In this research, RF source localization using fingerprinting (a technique involving matching to a known database) with signal autocorrelations is explored. A Gaussian-process-based method for autocorrelation-based fingerprinting is proposed. Performance of this method is evaluated using a ray-tracing-based simulation of an indoor environment.

ACKNOWLEDGMENTS

This thesis has been a great undertaking for me, and it is humbling to think of how many hands have supported me throughout his process. I don't know that it is possible to address everyone who has helped, or fully capture the difference they have made.

I am of course grateful for the support of my family, for my father and mother, for their encouragement and advice, my older brother and his wife, who were busy but still provided support, for my older sister and her husband, for talking through my difficulties with me and additional practical assistance, my younger sister and her husband, for helping me with writing and their companionship, and my younger brother for always being available to do things online together.

I was fortunate to work with and among many other exceptional students, who were companions as I fought through this work. In particular, I would like to thank Madi, my original research partner and a great support and friend, Colton, Thomas, Connor, and other lab members, who were all sources of comradery and support during the time we shared. You all seemed to believe I would eventually make it.

I am grateful for my other friends through the years, like Ryan and friends, who never forgot me, and helped make this long period bearable. I of course need acknowledge my roommate Nathan, who listened to all of my irritations, and made sure I took care of myself. There have been countless others, but I would lastly mention Sterling and Steven, who were there during some of the most difficult of moments.

It was an honor working with the Laboratory for Telecommunication Sciences; I am grateful for their support and for the great people there. I would also like to thank Nate and Austin from Autonomous Solutions, who were flexible with a complicated work situation and offered additional help and counsel, for far longer than they should have needed to.

Finally, I am grateful to the professors who have had taught me here, and especially to my graduate advisor Dr. Todd Moon, who was respectful and encouraging even when progress was slow, and to whom I owe so much of what I have learned.

Joseph L. Ipson

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction and Literature Review

1.1 Source Localization methods and challenges

The problem of "where is that signal coming from?" for radio frequency (RF) signals is well known, and has only become more important as wireless infrastructure has continued to develop. The problem of RF source localization (also referred to by the related term, RF geolocation) is the problem of identifying the location from which a particular RF signal is emitted. This problem is of practical interest whenever the location of some object is needed and that object is emitting an RF signal. This may include something that has been lost, such as a crashed vehicle or company asset. It might also be needed to locate and silence a signal which is interfering with the operation of another system. It is also relevant to many military situations, where it may be important to know the location of both allied and adversarial entities. In these differing situations, the resources and effectiveness of particular methods may vary substantially, motivating the use of a range of approaches. One such situation is depicted in Figure 1.1, where a single receiver is used to identify the position of a transmitter in an indoor environment.

1.1.1 Traditional localization methods

A wide variety of methods exist to perform source localization. These methods have different receiver requirements, and rely on different properties of the transmitted signal. Some of the most common methods are described by So [1] and Li et al. [2]. One of most basic methods, known as received signal strength (RSS), relies on measured signal power, utilizing the simple fact that signal power declines with distance in a regular fashion. This method is cheap and easy to implement, but generally fails to give high location precision. Another common method, called time-of-arrival (TOA) is by measuring the time taken to

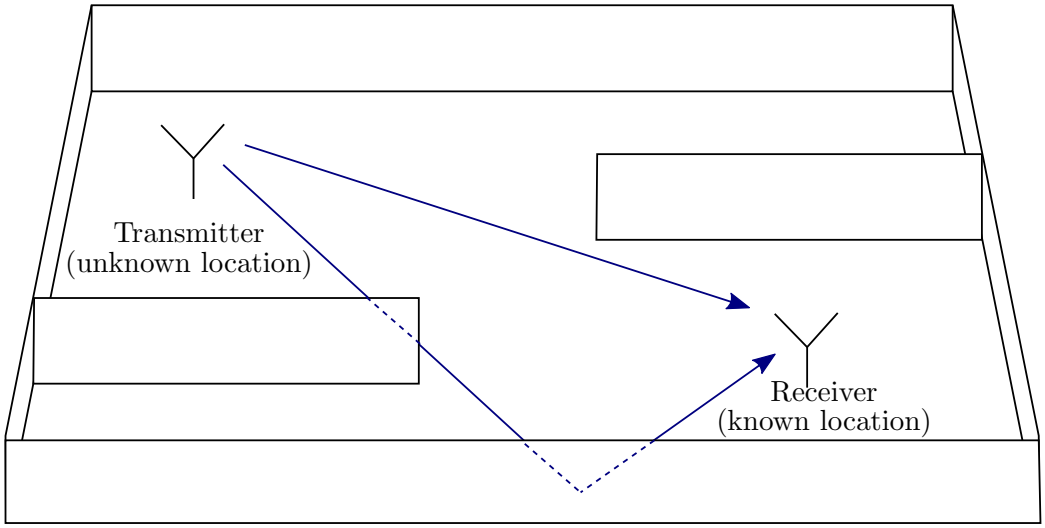


Figure 1.1: An illustration of a particular RF localization problem, where the signal at a single receiver of known location is used to attempt to identify the position of a transmitter at an unknown location in an indoor environment. This is the primary localization scenario discussed in this research.

reach the receiving antennas. This method can give very precise position estimates, but relies on time synchronization and adequate signal bandwidth, which are more difficult to achieve. Similarly, time-difference-of-arrival (TDOA) uses the difference between signal arrival times at multiple receiving antennas. It is similar to TOA, with the advantage of not requiring transmitter-to-receiver timing synchronization, but the disadvantage of requiring more receivers which must themselves still be synchronized with each other. A third method that is frequently employed is angle-of-arrival (AOA), which looks at phase differences at multiple nearby antenna to identify the direction of the source signal. When multiple arrays are used, this method also produces good precision, but requires more advanced antenna arrays which are suited for the frequency range of interest. A summary of the most common localization methods are given in Table 1.1. While these three methods are most prevalent in RF source localization, other methods, like those using signal frequency changes and those using channel information have also been developed.

1.1.2 Localization methods for unknown signals

In some situations the RF signal is designed with time stamps or other information

Table 1.1: Some common methods for RF source localization, derived from So [1]

Method	Advantages	Disadvantages
Received signal strength (RSS)	• Easy to implement • Inexpensive hardware • Usable in many situations	• Low precision • Performance degrades in some environments
Time-of-arrival (TOA)	• High precision	• Requires transmitter-to-receiver time synchronization • Dependent on signal bandwidth
Time-difference-of-arrival (TDOA)	• High precision	• Requires time-synchronized receivers • Requires more receivers than TOA • Dependent on signal bandwidth
Angle-of-arrival (AOA)	• Medium precision • Can be effectively combined with other methods	• Requires multiple or array of receivers • Receiver array must be time and phase synchronized • Antenna configuration required depends on frequency of signal

which can be leveraged by the receivers to locate the source. In many circumstances, however, the source signal is not known beforehand, and does not contain extractable timing information. This is often the case when the RF source of interest is hostile or was simply not designed with localization in mind. Under these conditions, the RSS method requires adjustment to use power differences, becoming differential received signal strength (DRSS).

Methods for DRSS localization are explored in work by Whiting et al. [3] and also included as modifications of RSS methods, as in Martin et al. in [4]. Likewise TDOA methods must be used instead of TOA, since transmitter-to-receiver synchronization is generally not possible. Techniques for TDOA are discussed in many publications, such as the work by Comsa [5] and Li et al. [2]. Similarly, some less common method types, such as channel or frequency methods, are more difficult, but still viable for unknown signals. For example the work by Hall et al. [6], can be used without knowing the transmitted signal beforehand. Angle-of-arrival methods do not necessarily rely on prior signal information, and can therefore be used with little or no modification in most cases.

1.1.3 Localization in indoor/multipath environments

Another challenge within RF source localization is coping with the effects of reflections and multiple signal paths, particularly in indoor environments. Localization in an indoor environment is depicted in Figure 1.1. The multiple arrivals often constructively and destructively interfere with each other, leading to changes in received signal power in different locations, an effect known as fading. Fading can reduce RSS/DRSS based location estimate accuracy. The multiple arrivals also make it difficult to precisely identify time-delays in TOA and TDOA, which can distort distance estimates. Likewise, AOA methods will detect the reflected paths' arrival directions as well, which may obscure the actual direction to the signal source. These timing and angle errors are particularly severe when the shortest path (called the direct path) is weaker than a reflected path. This is especially true under no-line-of-sight (NLOS) conditions, where the direct path is not detected at all by the receiver. In contrast to these more traditional methods, channel based methods actually leverage multiple arrivals to improve localization ability. A significant amount of research effort continues to be performed in addressing multipath problems in source localization. Pahlavan et al. provide a good summary of the challenges in this area in [7], though this work emphasizes cooperative localization systems. Some research focuses on using identifying and mitigating multipath and NLOS conditions on traditional methods, such as Saeed et al. [8–10].

1.1.4 Single receiver or unsyncronized receivers

A third major challenge in localization is limitations in available receiver hardware. Whether due to budgetary constraints, space limitations or other restrictions, extensive receiver systems are not always practical. The time-syncronized systems required for TOA/TDOA are significantly more expensive and difficult to set up. In some circumstances it may be desirable to perform localization with only a single receiver. None of the more traditional approaches discussed can uniquely resolve a position in this case, unless a hybrid method is used. Unlike these methods, channel information may be sufficiently rich to uniquely identify position from a single receiver measurement. Research for such single receiver localization systems has been carried out by Hall et al. in [6, 11] and O'connor et al. [12], which both use channel information.

1.2 Channel based localization methods

In signal processing, a channel describes the interface between an original source signal and what is collected by the receiver. In RF propagation situations, this becomes the total of all effects and changes which an RF signal undergoes as it is passed through a transmission system, through an environment, and finally through a receiver system. This can include a variety of effects, but can be modeled primarily as a linear system with additive noise. The linear system can be fully described by its response to an impulse function, which is known as the channel impulse response (CIR). The impulse response (and correspondingly the linear system) is directly related to the environmental features between a transmitter and receiver. To leverage the properties of the channel for localization, it is necessary to be able to estimate this linear system or some of its properties from what is measured by the receiver system. At this point, statistics or prior data can be used with the identified linear system to estimate the position of the RF source.

1.2.1 Utilizing channel information in received signals

If appropriate conditions are met, the linear system describing the channel can be estimated from what is measured by the receiver system. If the signal is known beforehand,

the channel can be estimated directly through deconvolution with the source signal. When source signal is not known, this becomes the more difficult problem of blind channel estimation. In this case, a variety of methods may be applied, depending on the type of signal and available receiver configuration. Many of these methods rely on multiple receivers, and can become more difficult if the receivers are not synchronized. A good summary of blind channel identification methods can be found by Tong and Perreau in [13]. Some methods even exist for estimation of the channel from a single receiver (see Darsena et al. [14]), but these methods often rely on other assumptions about the signal of interest. All channel identification methods are also limited to some degree by the bandwidth of the transmitted signal, which effects how the linear channel system and signal interact.

Channel information is utilized in a variety of methods to perform localization. In some cases the transmitted signal is used to estimate this information, as is done by Zhao et al. in [15]. Other cases perform blind channel identification, such as Hall et al. [6], and Rose et al. [16]. Another example using channel information for source localization is Nerguizian et al. [17], which focused on mitigating variation in channel parameters over time, another challenge in channel localization methods.

The effects of the linear channel are also embedded in the received signal power spectrum, or similarly, the signal autocorrelation. If the shape of the source signal spectrum is fairly flat in frequency, the received signal power spectrum will be similar to the spectral shape of the linear system. If only one received signal is available, this can be used in place of the linear system itself, missing only phase information. Channel identification is still possible by using the autocorrelation function and performing phase reconstruction, which is done by Baykal in [18]. Related methods for phase reconstruction are also summarized by Jaganathan et al. [19].Phase construction is often a difficult problem, so this approach may not always be effective. This received power spectral density (PSD), or equivalently the received signal autocorrelation function does not contain as much information as the full linear channel, but may in some cases still be usable for localization.

1.2.2 Estimating position from channel information

The traditional source localization methods rely on directly estimating a distance or direction relationship, possibly with the assistance of a statistical model. Channel methods do not have that luxury, as the relationship between the linear channel model and the source location is much more complicated and environment dependent. As pointed out by Qi et al. in [20], channel information is not necessarily useful without a model or additional environmental information. Some statistical models provide a statistical relationship between distance and paths, but this relationship is complicated and may only have weak relationship. For example the statistical channel model by Pahlavan in [21] allows for the potential of distance estimation from channel information, but the illustrative plots suggest only a weak relationship. The author does not even suggest this as a possible use for the model. Qi does describe how statistical models of NLOS components for an environment may be used [20], but again this requires more environmental information. This weakness motivates more data-driven and machine learning methods, for estimation source location from the channel.

1.2.3 Data-driven methods and fingerprinting

Because of the difficulty of fitting a relationship between the linear channel model or power spectrum and the source location, data-driven methods such as machine learning or fingerprinting are employed. Each of these methods involves producing a set of data which contains the channel information coupled with a corresponding source location. From this dataset, one can try to train a function which maps channel information to a position estimate. Alternatively, if the dataset contains positions throughout the region of interest for source localization, one can simply compare received channel information from a signal of interest to the dataset, choosing the location which matches the CIR most closely. This method is referred to as fingerprinting. Note however, that both of these methods are not actually very different, as fingerprinting is simply a way of creating a channel-to-position function.

1.2.4 Neural nets and current developments in fingerprinting methods

Some of the more recent work in the field of RF source localization is the use of deep learning or artificial neural networks (ANN). These may by applied using received signal strength as with as Nerguizian et al. [22], o they may use channel information like that of Wang et al. [23]. Deep learning still relies on collection of reference measurements and can still be considered a form of fingerprinting approach. Other methods continue to stick to standard machine learning methods, such a weighted K-nearest neighbor used by Hall et al. [6] or Zhao et al. [15]. The most comparable work to this research is likely that of Hall et al. in [11] and [6], which also pursues localization in a noncooperative context using fingerprinting and a single receiver. These methods however rely on a vector sensor, which provides for increased richness of available channel data.

1.3 Thesis topic and objectives

The goal of this research is to investigate RF source localization by using fingerprinting on received signal autocorrelations, particularly for signals with relatively limited bandwidth. The objectives of this thesis are as follows.

1. Outline the basic theory required to understand and carry out RF source localization using autocorrelation-based fingerprinting.

2. Develop more algorithms which can be used to perform RF source localization using autocorrelation-based fingerprinting.

3. Characterize the performance of the developed localization algorithms using simulated data.

4. Evaluate what bandwidth, frequency, and other requirements must be met for the use of these localization methods.

5. Demonstrate viability of methods by implementing them with physical hardware.

1.3.1 Thesis structure

The remaining thesis content will be proceed as follows. This method is used to generate some of the results of this thesis. Chapter 2 explains the basics of fingerprinting for source localization. Chapter 3 discusses methods and algorithms for using fingerprinting on autocorrelation functions. Chapter 4 describes the performance of these localization methods under different conditions. The primary results of this thesis are contained in Chapter 3 and Chapter 4. Chapter 5 will describe the appropriate RF system and propagation theory which underlies indoor propagation modelling used in this work. Chapter 6 describes a basic ray-tracing method using the theory developed in Chapter 5. Results and conclusions are summarized in Chapter 7. Finally, additional supplementary material are also available in Appendix A and code listings are included in Appendix B.

CHAPTER 2

Overview of Fingerprinting-Based Source Localization

2.1 Introduction and background

In source localization, fingerprinting is the process of identifying a source location by matching a received signal's features with a reference database which links known features to respective source locations. This name follows from the forensics practice of matching biometric information found at a crime scene (namely, a fingerprint) and comparing it to a database of that same biometric information to match individuals to a crime.

This differs from more traditional source localization methods, where certain properties of the received data (such as the time-of-arrival, or received signal strength) which have knwown relationships to position are used. In fingerprinting, the features of the signal, also called the "signature", can be any combination of calculable signal properties that has information pertaining to position. This could include these same traditional measurements, like received signal strength, time-of-arrival, and angle-of-arrival, possibly in combination. It is also possible to utilize signal characteristics which can be otherwise difficult to use in estimating position, such as an estimated CIR, as done by Sousa et al. in [24]. Similarly, it is possible to use signal autocorrelation for fingerprinting; this method in particular will be discussed in detail in this thesis.

As localization is essentially trying to find a function that takes "received data" as input and produces "location" as output, fingerprinting can also be consiered a machine learning method. It creates the localization function by using the reference measurements as "training data" to produce a position from the calculated signature.

2.1.1 Advantages of fingerprinting methods

Fingerprinting is generally more complicated than traditional approaches for source

localization, such as typical RSS, TOA/TDOA, and AOA approaches. The need to produce a reference map can be a tedious step which is unnecessary for these other approaches, and location estimation is not necessarily faster. With that in mind, there are three primary motivations for which one would favor a fingerprinting approach. These are: (1) more fully accounting for environmental effects, (2) employing signal properties which are not otherwise easily used in localization, and (3) being able to accomplish the previous advantages with only a few unsycronized receivers or even a single receiver.

First, a fingerprinting approach can account directly for variations in an environment that could only be modeled statistically in other approaches. For example, in RSS source localization, most environmental fading is just modelled as an unknown statistical error on the power measurement or corresponding loss coefficient, which can introduce significant localization error. When fingerprinting is used, environmental fading effects are included within the reference map itself, so these effects may not lead to the same error.

Second, fingerprinting can be used on signal properties which do not have a simple relationship to position, such as the CIR. Fingerprinting using signal autocorrelation, the primary topic of this thesis, fits into this category. While some statistical models suggest a relation between multipath components and transmitter-to-receiver distance, it is more accepted that utilizing these components without environmental knowledge is not effective (see Qi et al. [20]). Such relationships also assume multipath components can be estimated from the autocorrelation or CIR, which is likewise a difficult problem. However, usage of CIR or autocorrelation in fingerprinting can be done directly using the methods described in this chapter and Chapter 3.

Third, localization using this fingerprinting fundamentally only needs one receiver, and can use receivers which are not time-syncronized when multiple receivers are used. While it may not always be possible to successfully identify a transmitter's position using a single receiver, the method does not fundamentally require multiple receivers to function. If more receivers are needed, these receivers only need coarse time-synchronation, possibly on the order of hundreds of milliseconds, compared to the nanosecond synchronation required for

TOA/TDOA methods. How many receivers are required will depend on the bandwidth and other factors (see results in Chapter 4), but the lack of need for synchronized systems may offer substantial savings in system cost.

2.1.2 Summary of fingerprinting process

The fingerprinting process can be divided into two phases: a preliminary, offline data collection phase, and an online localization phase. The first phase, the offline data collection or training phase, is where data is collected from the target environment and preparatory processes are performed. Fingerprinting source-localization relies upon a set of reference measurements where signal features are associated with a source position. This map can be represented as an array of M tuples of the form $(\boldsymbol{r}_m, \boldsymbol{x}_m)$ where $\boldsymbol{r}_m$ is the signature calculated from a signal which is received by a transmission from a known location $\boldsymbol{x}_m$. Note that $\boldsymbol{r}_m$ can be either a scalar or vector quantity.

The second phase, the online localization phase, is where a transmitter can be localized by leveraging the resources obtained in the offline phase. In this phase, a signal-of-interest $y_{\text{soi}}(n)$ will be processed to identify its source location $\boldsymbol{z}$. Upon receiving this signal, the signature will be calculated and preprocessing steps will be performed on this signal in the same manner that the reference measurements were processed. After this is done, a score value (such as a distance or likelihood) is calculated for each of a set of candidate transmitter positions. Finally, the location with the optimal score is chosen as the estimated transmitted location.

These steps can be summarized as:

1. Offline training phase
    a. Collect measurements of signals from set of locations $\{\boldsymbol{x}_m | m = 0 \dots M-1\}$, signatures are calculated, processed and stored as $\{\boldsymbol{r}_m | m = 0 \dots M-1\}$
    b. Perform any additional processing to prepare for online phase, such as fitting ditributions (Section 2.3).
2. Online localization phase

a. Process measurement and calculate signature $\boldsymbol{r}_{\text{rec}}$ for signal-of-interest.

b. Calculate score for each candidate location $\boldsymbol{z}$, by using information from reference measurements

c. Estimated location is given by candidate point which maximizes score (or minimizes error)

This process is depicted in Figure 2.1. More details regarding these two phases will be discused in the following sections.

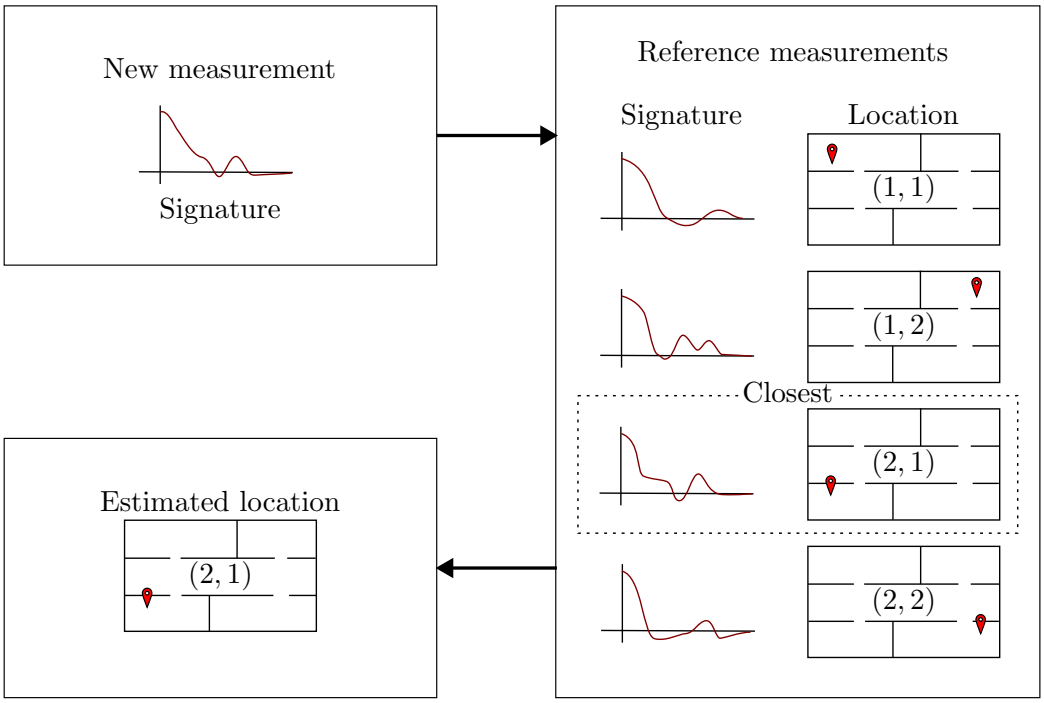


Figure 2.1: Summary of fingerprinting algorithm. Localization is performed by comparing a measured signature with a set of reference measurements, which are each associated with a source location. The "closest" signature is found, and the estimated source location is given by that associated reference location.

2.2 Offline training phase - collecting reference measurements

The signature (such as the channel impulse response or autocorrelation) for each measurement is calculated. Any necessary preprocessing steps are performed such as those

discussed in 3.1. This may include normalization, feature extraction, etc. Additional processing steps may be performed to prepare for localization, such as the creation of a posterior distribution, as done in the Gaussian processes method described in Section 3.2.

2.2.1 Reference measurements and signature properties

Being able to produce a set of reference measurements is necessary for a fingerprinting method. This is typically done by transmitting test signals (often a known white noise signal) from a set of locations $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{M-1}$ throughout the region of interest, calculating the relevant signature $\boldsymbol{r}_m$, and recording this signature for each position. Alternatively, a model of the environment can be used to simulate this process, and received signatures may be generated for each position instead.

It is neither necessary, nor possible for this set of reference measurements, or reference map, to contain every possible emitter position. However is important to produce a map that has enough points to resolve emitter locations in the region of interest. The necessary resolution depends on the behavior of the signature, and is discussed further in Section 2.2.3.

When using simulation, if the environment is not modeled with enough fidelity, the generated map will not be effective. Ray-tracing or other propagation modeling may be sufficient to be useful, but modeling accuracies could cause map defects which then hurt performance. With some methods, using simulated data to supplement physically collected data may be an option, as done by Sousa et al. [24].

On the other hand, if the map is created through physical testing, it is necesary to take an emitter with known location, and repeatedly sample the signature for that emitter at the receiver at different at each map location. Creating the map this way requires free movement within the region of interest and mobile emitter hardware. The process can also be time intensive, particularly if the region is large, and if high map resolution is necessary. If these difficulties can be dealt with, this method should produce better fingerprinting performance than using simulation-produced measurements.

2.2.2 Signature properties

In order for a signal property to be a useful signature for fingerprinting it must foremost have some natural relationship with transmitter position. In the case of RSS and TOA/TDOA, the relationship between transmitter position and the signal property can often be expressed as a direct relationship to distance. For channel information approaches, the relationship is not as easily stated, but it is intuitive that the shape of these functions will vary to some extent with transmitter position. This section further describes some other properties which can affect fingerprinting performance.

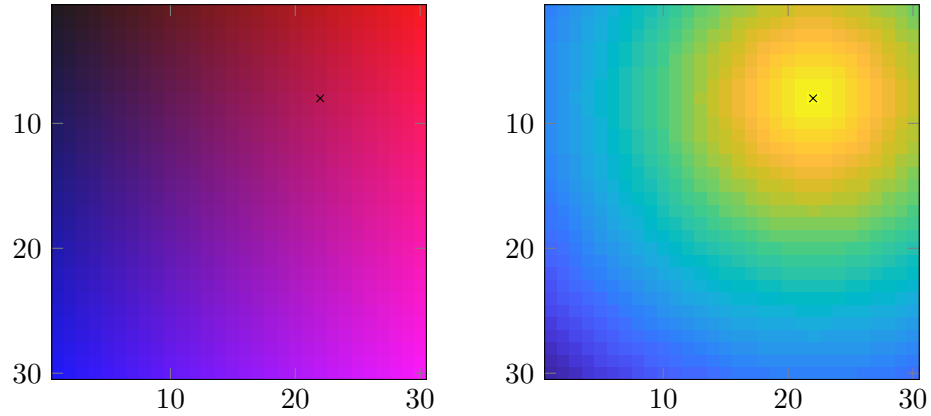


Figure 2.2: (Left) A representation of a reference map with good smoothness and uniqueness. The axes represent the physical location and the colors represent signature value. In this figure, the signature and position are smoothly related. (Right) A simple distance-based score is shown, which compares an arbitrary point's signature with the particular point denoted with an x in the left image. Here the score is highest around the correct point, a property which means that even imperfect localization will still yield a nearly the correct location.

2.2.3 Smoothness

Smoothness is the propensity of a signature from one transmit location to be "similar" to a signature from a transmit location nearby. An illustration of reference maps with and without good smoothness are shown in Figure 2.2 and Figure 2.3 respectively.

Signature smoothness has a direct effect on the reference map resolution needed to perform effective estimation. This can only be ensured for any potential transmit location

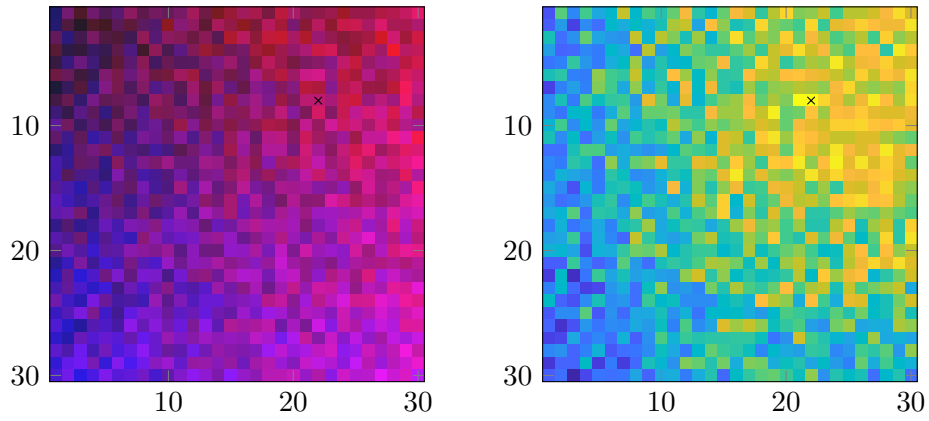


Figure 2.3: (Left) A representation of a reference map with lower smoothness. The axes represent the physical location and the colors represent signature value. In this figure, the signature and position are well related, but sometimes abrupt changes. (Right) A simple distance-based score is shown, which compares an arbitrary point's signature with the particular point denoted with an x in the left image. Here the score is still high around the correct point, but the score can be significantly changed with a slight change in position, possibly leading to localization errors.

with a sufficiently fine resolution map. This can be viewed as a form of spatial Nyquist criterion, where the rate of variation in physical space effects the degree of sampling necessary. If a signature does not have a sufficient quality of smoothness, fingerprinting will not be effective.

2.2.4 Uniqueness

Uniqueness is the property of a signature to only obtain a particular value in a single location. This is contradicted when the signature looks highly similar in two locations which are not close to each other. This is illustrated in Figure 2.4.

While arguably less critical then smoothness, uniqueness issues can still cause large location errors, as an optimal score can occur for a transmit position and reference position which are far apart. Uniqueness is closely related to the dimensionality or "richness" of nonredundant information within a typical signature. Uniqueness can be improved by obtaining more informative measurements (for example by using a receiver with directionality or polarization information), or by obtaining more measurements. This can be done either by using multiple receivers or, in the case of a moving transmitter, multiple measurements

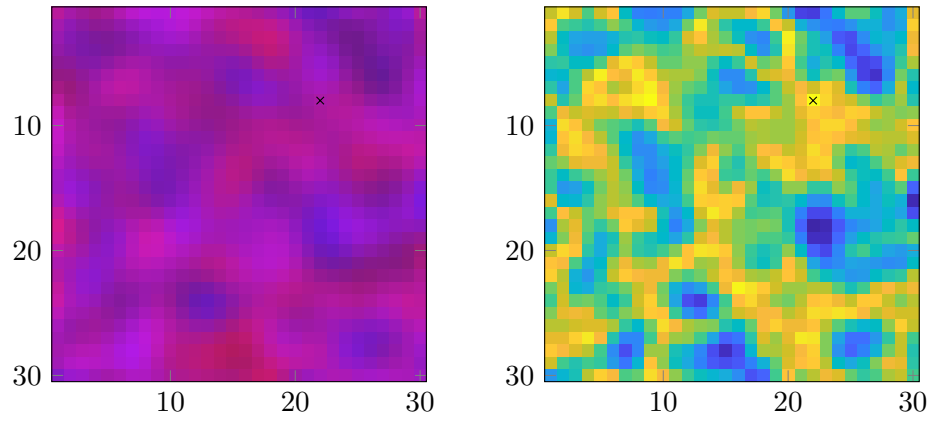


Figure 2.4: (Left) A representation of a reference map with poor uniqueness properties. The axes represent the physical location and the colors represent signature value. In this figure, the signature varies smoothly, but similar signatures occur in highly varying locations. (Right) A simple distance-based score is shown, which compares an arbitrary point's signature with the particular point denoted with an x in the left image. Here the score is still high at the correct point and in a region surrounding it, but there are multiple regions with a high score, leading to an inability to fully resolve the location by using this score.

taken over time. Severe uniqueness problems can lead to high distance errors in localization, and must be addressed for effective fingerprinting.

2.2.5 Time-stationarity and reproducibility

A major concern for fingerprinting approaches pertains to whether signals transmitted from the same position at different times produce essentially the same signature. In practice, all propagation environments are time-varying to some degree. This may be due to the physical motion of people or objects, building renovations, or any other changes in environment over time. Most useful localization signatures will be influenced by these changes, even if only slightly. This effect is in many ways analagous to smoothness, except that it is instead sensitivity to slight environmental changes over time rather than position.

If signatures are affected too severely by a time-varying environment, reliable fingerprinting may not be possible. If the time-varying effects are sufficiently slow, then they may be solved by producing a new reference map periodically. This is not desirable in general, due to previously mentioned difficulties in producing reference maps. Alternatively, if the initial set of reference measurements includes redundant measurements taken during differ-

ent times or conditions, these additional measurements may be usable to make the system more robust to such changes. An approach like this is used by Nerguizian et al. [17]. It may also be possible to design systems which are capable of a degree of updating the reference map while live, but this may require additional hardware and is beyond the scope of this research.

2.3 Online localization phase — basic fingerprinting

The most straightforward method of fingerprinting is to choose the position corresponding to the reference measurement that is the "best match" for the signal-of-interest. This can be likened to a one-nearest-neighbor approach in machine learning. First, in order to be fairly compared to the reference measurements, the signal-of-interest must undergo the same method of calculation and preprocessing as the reference measurements. Then it is necessary to calculate some form of "score" for how closely matched the signal-of-interest is with any particular reference measurement. This score can either be based on a simple vector distance or it can be developed probabilistically as a likelihood. With $\boldsymbol{x}_{\text{opt}}$ as the optimal estimated position, $\boldsymbol{x}_m$ as the position from the set of reference measurements with index m, $\boldsymbol{r}_{\text{rec}}$ as the signature from the signal being localized, and $\boldsymbol{r}_m$ as the reference measurement with index m, this can be written as

$$\boldsymbol{x}_{\text{opt}} = \boldsymbol{x}_{m^*}, \text{ where } m^* = \arg\min_m d(\boldsymbol{r}_{\text{rec}}, \boldsymbol{r}_m), \tag{2.1}$$

where $d(\boldsymbol{r}_i, \boldsymbol{r}_j)$ is a metric which mathematically expresses "closeness" for the signature used. A simple example would be an L_2 norm distance, so $d(\boldsymbol{r}_i, \boldsymbol{r}_j) = \|\boldsymbol{r}_i - \boldsymbol{r}_j\|^2$. However, since the selection of metric can directly affect fingerprinting performance, this may not necessarily be the right choice.

If a single location estimate is desired, one simply selects the location with the highest final score. In most cases, this is simply a matter of searching over the entirety of the reference map for the minimum. Alternatively, a spread of likely positions can be given as the final output, depending on the final system to be used. Particularly, if a likelihood or

posterior probability is used as a score, this information can be naturally integrated into an algorithm using multiple sensors or methods. Such a method also allows for the target to be tracked over time using a particle filter, hidden Markov model, or similar Bayesian tracking method. For an example of the use of a particle filter for a similar system see Ferris et al. [25].

Maximum-likelihood estimation

Due to being more interpretable and more convenient for integrating with other methods, a likelihood is often the preferable function for use as a score. The likelihood function describes the probability some set of observations could occur, if a particular model and model parameters had produced it. Thus maximum-likelihood estimation attempts to choose a model, or more typically, model parameters, such that the observation is most likely to have been produced by the model. In the maximum-likelihood formulation of source localization based on a signal signature can be written as

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} p(\boldsymbol{r}_{\text{rec}}|\boldsymbol{z} = \boldsymbol{x}), \tag{2.2}$$

which is analogous to the previous (2.1). This formulation requires using a statistical model. A reasonable model using Gaussian methods is described in Section 3.2. However a simpler method can be found by fitting a simple distribution to the L_2 distance associated with an accurate position match. That is, let $\boldsymbol{r}$ be the signature associated with a transmit location $\boldsymbol{x}$, then let $\boldsymbol{x}_\epsilon$ be the signature associated with some position $\boldsymbol{x}_\epsilon$ which is withine some small distance ϵ of $\boldsymbol{x}$ (so $\|\boldsymbol{x} - \boldsymbol{x}_\epsilon\| < \epsilon$). Then one can model the distance in signature associated with these two points $d(\boldsymbol{r}, \boldsymbol{r}_\epsilon)$ as a random variable D_ϵ. If a distribution is fit or chosen for D_ϵ, one can express the probability in (2.1) as

$$p(\boldsymbol{r}_{\text{rec}}|\boldsymbol{z} = \boldsymbol{x}_m) = p(D_\epsilon = d(\boldsymbol{r}_m, \boldsymbol{r}_{\text{rec}})) \tag{2.3}$$

noting that if $\boldsymbol{z} = \boldsymbol{x}$ then $\|\boldsymbol{z} - \boldsymbol{x}\| < \epsilon$, so the error distribution applies. To select a

distribution for the error D_ϵ, one can look at the signature distance within the reference measurements $d(\boldsymbol{r}_i, \boldsymbol{r}_j)$ with respect to the physical distance $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|$. A histogram for this signature distance for reference measurements in close proximity is shown in Figure 2.5.

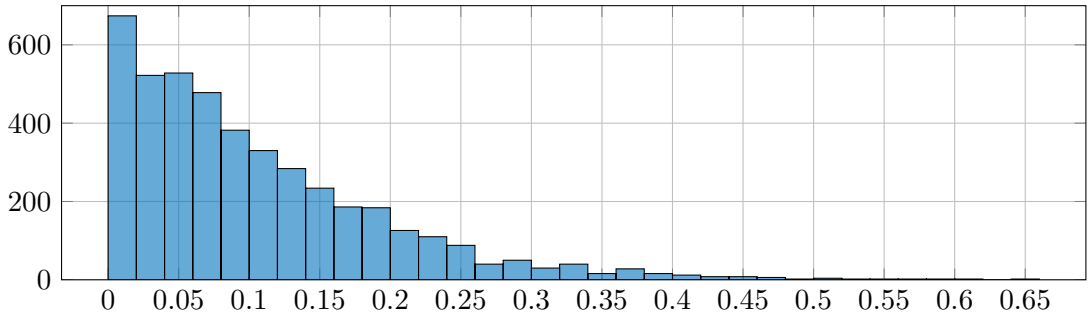


Figure 2.5: Histogram of l_2 distance between signatures for positions within 1.2 m, using 128MHz bandwidth

An exponential distribution (or any other distribution which fits the data in Figure 2.5) can be used as the distribution for D_ϵ.

Posterior distribution

In some cases it can be useful to go from a likelihood to a posterior distribution. This can be done in a fairly straightforward manner by using Bayes rule:

From this, a posterior distribution can be set up on grid of reference points using Bayes rule, giving

$$p(\boldsymbol{z} = \boldsymbol{x}_m | \boldsymbol{r}_{\text{rec}}) = \frac{p(\boldsymbol{r}_{\text{rec}} | \boldsymbol{z} = \boldsymbol{x}_m) p(\boldsymbol{z} = \boldsymbol{x}_m)}{\sum_{m=0}^{M} p(\boldsymbol{r}_{\text{rec}} | \boldsymbol{z} = \boldsymbol{x}_m) p(\boldsymbol{z} = \boldsymbol{x}_m)} \tag{2.4}$$

If no prior emitter location information is available, assuming a uniform distribution gives simply the probability distribution:

$$p(\boldsymbol{z} = \boldsymbol{x}_m | \boldsymbol{r}_{\text{rec}}) = \frac{p(\boldsymbol{r}_{\text{rec}} | \boldsymbol{z} = \boldsymbol{x}_m)}{\sum_{m=0}^{M} p(\boldsymbol{r}_{\text{rec}} | \boldsymbol{z} = \boldsymbol{x}_m)} \tag{2.5}$$

2.3.1 Multiple receivers

When multiple detectors are used, these multiple measurements can be used as vector components in a single signature, or the results of multiple fingerprinting estimations can be combined. To combine vector measurements into a single signature, one can simply create a new signature vector

$$\tilde{\boldsymbol{r}} = \begin{bmatrix} \boldsymbol{r}_{\mathrm{RX}0} \\ \boldsymbol{r}_{\mathrm{RX}1} \\ \vdots \\ \boldsymbol{r}_{\mathrm{RX}N-1} \end{bmatrix} \tag{2.6}$$

where $\boldsymbol{r}_{\mathrm{RX}n}$ is the signature from the nth receiver. The typical procedure would be used in the offline training and online localization phases, except using this augmented signature instead of the individual one. Alternatively, the signatures from each receiver can be handled separately, and only combined when calculating a score in the localization phase. In this case, the fingerprinting process is duplicated and handled individually for each receiver, but during localization a combined score is calculated using each individual score. This is most easily done if a maximum-likelihood approach is used, and signature information is assumed to be independent for each receiver. In this case the joint score (likelihood) is simply the product of the individual likelihoods

$$p(\boldsymbol{z} = \boldsymbol{x}_m | \{\boldsymbol{r}_{\mathrm{recRX}n}\}_{n=0}^{N-1}) = \prod_{n=0}^{N-1} p(\boldsymbol{z} = \boldsymbol{x}_m | \boldsymbol{r}_{\mathrm{recRX}n}). \tag{2.7}$$

If a likelihood approach is taken, this is often preferred to using an augmented signature vector, as it will typically be less computationally expensive.

2.3.2 Fingerprinting and deep learning

Fingerprinting is inherently a data-driven approach, and is thus closely related to machine learning methods. The reference measurements can be viewed as a site-specific set of training data, with the signatures $\boldsymbol{r}_m$ as the system input and associated locations as the output. This means that many of the typical machine-learning tools, including deep

learning, can be applied. These methods can either be applied to the localization problem as a whole, or it can be applied to specific parts of the problem (as in how autoencoders are used for feature extraction by Hall et. al in [11]). This research is not focused on deep learning approaches, so they will not be explored in detail here. It is worth noting that the applicability of deep learning depends on the availibility of sufficientlly rich data, and so these will be better suited to cases where larger sets of reference measurements are available. Additionally, many of the principles discussed in this section are still relevant to fingerprinting performance, regardless of whether ANNs or other deep learning methods are applied.

2.4 Tracking a moving source

If the emitter is moving, sequential Bayesian approaches can be applied. In the case of a discrete fingerprinting approach, this can be done by treating the moving emitter as a hidden Markov model. In this model, the emitter is modelled to move randomly to nearby positions at set time intervals. The emitter location constitutes the state of the Markov model. The autocorrelation is estimated over these distinct time intervals, and corresponds to the measured Markov model output.

2.4.1 Hidden Markov model

For a good resource on the basics of hidden Markov models, consult Rabiner [26], from which this summary is adapted. Further discussions and derivations can be found in Moon and Stirling [27].

A hidden Markov model consists of the following predetermined elements:

- a discrete set of M system states $\{l_i\}_{i=1}^{M}$;
- A number T of discrete-time steps;
- Initial state probabilities $\{\pi_i\}_{i=1}^{M}$, with $\pi_i = p(q_0 = l_i)$;
- State transition probabilites $\{a_{ij}\}_{i=1,j=1}^{M,M}$, with $a_{ij} = p(q_{t+1} = l_j | q_t = l_i)$;

- A likelihood function for the observed system output, given a particular state $b_i(t) = p(y_i|q_t = l_i)$;

Additionally, these properties will be produced by the operation of the model:

- A sequence of T states the system takes on $q_0, \ldots, q_{T-1}$;
- A sequence of T observable outputs produced by the system $y_0, \ldots, y_{T-1}$;

Using the predetermined elements, these properties are produce using the following operation of a hidden Markov model:

1. Initial state: The Markov model begins at an initial state $q_0 = l_{m_0}$ where m_0 is chosen randomly with probabilities given by $p(q_0 = l_i) = \pi_i$
2. Initial output: An output y_0 is generated based on the state q_0 such that $p(y_i|q_t = l_i) = b_i(t)$.
3. State transition: A new state for $t + 1$ is randomly selected based on the previous state t, with probabilities $p(q_{t+1} = l_j|q_t = l_i) = a_{ij}$
4. Next output: A new output for $t+1$ is generated satisfying $p(y_i|q_{t+1} = l_i) = b_i(t+1)$ based on the updated state q_{t+1}
5. Repeat: Steps 3-4 are repeated until all states and outputs are generated for $t = 0, \ldots, T-1$

Note that this process describes the operation of a hidden Markov model, not how estimation is performed on such a model. This is the underlying structure that the data is assumed to follow, from which optimal estimation can be performed. In most cases, estimation is performed under the assumption that the observable system outputs $y_0, \ldots, y_{T-1}$ are known, while the system states $q_0, \ldots, q_{T-1}$ are not known. In order to perform estimation it is also necessary to have predetermined hidden Markov model elements on hand. These can be estimated (this is discussed in Rabiner [26]), but model parameters can also be chosen based on knowledge of the specific application.

2.4.2 States — transmitter position

In this application the states q_t are used to represent the location of the transmitter-of-interest at a given time t. As there are only a finite set of M states, a set of representative positions are selected throughout the region in which localization is to be performed. Note that this limits locations where a transmitter can be tracked, but this is actually already a limitation for fingerprinting-based localization systems, so this is not a large issue in this case. The state sequence represents the position over time, thus state transitions represent transmitter motion.

2.4.3 Transition probabilities — motion model

The motion model in this construction reflects how likely it is for an emitter in one location to move to an emitter in another location. This model reflects the fact that an emitter is unlikely to move great distances in a short period of time. A simple model can be constructed in the following manner:

$$a_{ij} = p(q_{t+1} = l_j | q_t = l_i) = \frac{\exp(-\frac{d(l_i,l_j)}{2\sigma_d})}{\sum_j \exp(-\frac{d(l_i,l_j)}{2\sigma_d})}, \tag{2.8}$$

with $d(l_i, l_j)$ as the physical distance between two emitter positions l_i and l_j, and a parameter σ_d which indicates scale. Note that this is simply a gaussian kernel fit to the distance, which is then normalized to be a probability distribution over the discrete set of locations l_j.

Performance can be improved if the distance $d(l_i, l_j)$ is adjusted to account for an inability for an emitter to move through walls, or if the realism of the motion model is otherwise improved. It is also possible to enhance the set of states to include motion in different directions, but this comes at a computational price. Non-discrete movement models are outside the capacity of a basic hidden Markov model. Such models may be more effective, but are not investigated in this work.

2.4.4 Likelihood probabilities - fingerprinting posterior

The likelihood probabilities reflect the information provided by the received signal

signature. These come from the likelihood of a particular position based on the signature, as discussed in Section 2.3:

$$b_i(t) = p(\boldsymbol{r}_{\text{rec}}(t)|\boldsymbol{z}(t) = \boldsymbol{x}_i), \tag{2.9}$$

Forward and backward probabilities

As an intermittent step in estimating states using a hidden Markov model, it is necessary to compute values known as forward and backward probabilities. Forward probabilites represent information from past states that can be used to predict future states and are written as $\alpha_i(t)$. Backware probabilites likewise represent information from later states that can be used to predict past states and are written as $\beta_i(t)$. They are defined and calculated iteratively as shown below.

Forward Probabilites:

1. Definition:

$$\alpha_i(t) = p(y_0, \ldots y_t, q_t = l_i), \quad t = 0, \ldots T-1 \tag{2.10}$$

2. Initialization:

$$\alpha_i(0) = \pi_i b_i(y_0), \quad i = 1 \ldots M \tag{2.11}$$

3. Induction:

$$\alpha_j(t+1) = \left[\sum_{i=1}^{M} \alpha_i(t) a_{ij}\right] b_i(y_{t+1}), \quad j = 1 \ldots M \tag{2.12}$$

Backward Probabilites:

1. Definition:

$$\beta_i(t) = p(y_{t+1}, \ldots y_T | q_t = l_i), \quad t = 0, \ldots T-1 \tag{2.13}$$

2. Initialization:

$$\beta_i(T-1) = 1, \quad i = 1 \ldots M \tag{2.14}$$

3. Induction:

$$\beta_i(t-1) = \sum_{j=1}^{M} \beta_j(t) a_{ij} b_j(y_t), \quad i = 1 \ldots M \tag{2.15}$$

2.4.5 Heatmap-over-time

Using the hidden Markov model a range of plausible locations and their probabilities can be found, which is often more useful than knowing the single most likely path. In this case one can calculate the posterior probability that the emitter was in location l_i at time t using the formula

$$p(q_t = l_i | y_0, \dots y_{T-1}) = \frac{\alpha_i(t)\beta_i(t)}{\sum_{i=1} \alpha_i(t)\beta_i(t)}, \tag{2.16}$$

where α and β are the forward and backward probabilities discussed in Section 2.4.4. Note that this method does not indicate path probabilities, but rather individual state probabilities, which may be less ideal for some applications. The algorithm to calculate the most likely complete path of states is given by Rabiner [26], and resembles the Viterbi alogorithm used in communications systems.

CHAPTER 3

Channel Vector Fingerprinting Methods

The methods discussed in this section here have been previously published by Ipson and Moon [28], they are included here with substantially more detail and illustration.

3.1 Vector preprocessing

Before performing localization using a signal autocorrelation or channel impulse response, it is necessary to perform some adjustments to make the resulting vectors suitable for fingerprinting localization. This typically includes some form of normalization to remove scaling effects, and may also include windowing. If the channel has complex values, it may be necessary to convert the vector to a real valued form, depending on the fingerprinting processing method. Finally, some methods may benefit from applying a transformation that produces a vector of features, rather than using the original vectors.

3.1.1 Isolating signal and calculating autocorrelation

Upon receiving a signal (either for offline reference data or during localization), some initial processing may need done to isolate the signal of interest and frequency shift the signal to baseband, as discussed in Chapter 5. After these steps are performed, the autocorrelation of the signal can be calculated as

$$r_y(k) = \frac{1}{N-k} \sum_{n=0}^{N-1} y(n) y^*(n-k). \tag{3.1}$$

This autocorrelation function $r_y(k)$ will then be adjusted as described in the remainder of this section. As the autocorrelation is conjugate symmetric, it is only necessary to use half of the autocorrelation function with either positive or negative indices. It is practical to

represent this as a finite vector, or

$$\boldsymbol{r}_y = \begin{bmatrix} r_y(0) & r_y(1) & \cdots & r_y(K) \end{bmatrix} \tag{3.2}$$

for some integer K, chosen such that elements of $r_y(k)$ are negligible when $k \geq K$.

3.1.2 Noise term removal

Another effect of taking the autocorrelation is that it concentrates receiver noise at $k = 0$ in the autocorrelation. By simply truncating the vector at this index (removing the zero-index element), a significant part of the effects of receiver noise can be removed. This noise will still contribute to the variance of the autocorrelation estimate when using a finite number of samples. However, the relevance of this effect diminishes as the number of samples used to calculate the autocorrelation increases. With this addition, the autocorrelation vector expression can be updated to

$$\boldsymbol{r}_+ = \begin{bmatrix} r_y(1) & r_y(2) & \cdots & r_y(K) \end{bmatrix}, \tag{3.3}$$

which excludes $r_y(0)$.

3.1.3 Normalization and windowing

It is often necessary to normalize channel vectors to avoid vector features being obscured by the scale or magnitude of the vector elements. While the relative scale between elements within the autocorrelation vector contains important information, the shared scale of elements of the vector as a whole is primarily a function of the received signal power. While received signal power is often useful in localization methods, it can entirely obscure the other channel vector features which are being compared for fingerprinting. Removing this power allows fingerprinting to focus on the channel "shape," which can be quite informative for position. If signal power is also being utilized for localization (a good idea wherever feasible) then the signal power can be incorporated in a different method, such as

by using separate fingerprinting on the signal power and multiplying the final likelihoods from power and channel information. For more information on considerations for using Gaussian processes, see Aravecchia and Messelodi [29] and Ferris et al. [25]. Normalization is done by dividing every element of the vector by a scaling value, usually the l_2 norm of this vector. This gives the normalized vector

$$\boldsymbol{r}_N = \frac{1}{\|\boldsymbol{r}_+\|}\boldsymbol{r}_+, \tag{3.4}$$

Additionally, since the autocorrelation estimate has higher variance with higher index values k, it is good practice to apply a window to this autocorrelation to diminish the influence of these values. For a causal discrete-time sequence with N nonzero elements, $N-k$ samples will be available to compute the kth autocorrelation term, as reflected in (3.1). This means that higher indices of k will be computed with less samples, increasing variance. Since the ideal values (as would be computed from an infinite sample) in the autocorrelation are typically small in magnitude, the estimated values can be dominated by the estimation variance. Hence, applying a window function to diminish the contribution of these values can avoid possible errors due to this variance. To address this, a triangle window is applied to limit effects of noisy estimates of the autocorrelation tails, so

$$\boldsymbol{r}_W = \left[\frac{K}{K}\boldsymbol{r}_N(1) \quad \frac{K-1}{K}\boldsymbol{r}_N(2) \quad \cdots \quad \frac{K-k}{K}\boldsymbol{r}_N(k) \quad \cdots \quad \frac{K-K+1}{K}\boldsymbol{r}_N(K)\right]. \tag{3.5}$$

3.1.4 Treatment of complex-valued vectors

Autocorrelation vectors are, in general, complex valued. Many approaches can be generalized in a way that supports complex values, such as by using the complex version of the l_2 norm. Alternatively, the imaginary part can be concatenated to form an extended real vector, leading to increased vector length, but otherwise no algorithm change. Another approach is to use the Fourier transform of the autocorrelation, which real valued from the autocorrelation symmetry. Another a simple method to address this, one can simply take the magnitude of each element. This last approach is used in this research, due to

its simplicity, and as preliminary testing did not suggest any performance benefit from the other methods. For a comparison on performance of some of these methods, see the results in Section 4.2.9. With this the preprocessed autocorrelation vectors become

$$\boldsymbol{r} = \left[\frac{K}{K}|\boldsymbol{r}_N(1)| \quad \frac{K-1}{K}|\boldsymbol{r}_N(2)| \quad \dots \quad \frac{K-k}{K}|\boldsymbol{r}_N(k)| \quad \dots \quad \frac{K-K+1}{K}|\boldsymbol{r}_N(K)| \right]. \tag{3.6}$$

This form of the autocorrelation vector will be used for both reference measurement vectors $\boldsymbol{r}_m$ and vectors for a signal-of-interest $\boldsymbol{r}_{\text{rec}}$.

3.1.5 Feature extraction

While many algorithms can be performed directly on the autocorrelation vector produced by the preprocessing steps discussed previously, it is often preferrable to calculate statistics or features from these vectors and use these instead. The method of extracting these features can be chosen to have more favorable properties for a given algorithm. Feature extraction can be seen as a mapping F from a raw data vector which is received through measurement to a feature vector which may be better suited for further use, expressed as

$$F : \boldsymbol{r} \mapsto \boldsymbol{q}. \tag{3.7}$$

It is desirable to choose F such that the features $\boldsymbol{q}$ have particular properties. This may include, but is not limited to, fitting a particular model, being more interpretable, not containing irrelevant information, being of lower dimension, and being statistically independent. While in general F may or may not be a linear mapping, restricting this to a linear operator greatly simplifies finding a suitable value for it. In this research, the reference measurements will be used to produce a suitable linear operator for F.

In the case of the Gaussian process model used in section 3.2, ideal features would be approximately independent, follow a Gaussian model and eliminate information which did not relate to position. However, determining ideal features is not an easy problem, so features attainable through a principal-component-analysis-like method are used in this

research.

An alternative method for feature extraction is posed by Nerguizian et al. [30]. This method may produce better features, but it has not been explored in combination with this research. As another option, features are extracted using an autoencoder by Hall et al. [6]. In this work features were produced using an singular-vector-based method similar to principal component analysis, due to it being straightforward and due to the approximate independence it gives, as described in Section 3.1.7.

3.1.6 Singular vector features

Principal components analysis is a method to separate out important, uncorrelated information from a data matrix. The first principal components are projected components of the original data vectors which optimally retain variance from the original vectors. These principal components are used in a variety of settings for producing features for machine learning (see Géron [31]). In this work, a method similar to Principal component analysis is used, where singular vectors from the reference measurements are used to select suitable linear features.

To obtain the features, one first removes the componentwise sample mean from the set of reference measurements, that is

$$\tilde{\boldsymbol{r}}_i = \boldsymbol{r}_i - \frac{1}{M}\sum_{m=0}^{M-1} \boldsymbol{r}_i \tag{3.8}$$

One can then form the resulting mean removed data matrix, and form its singular value decomposition.

$$R = \begin{bmatrix} \tilde{\boldsymbol{r}}_0 & \tilde{\boldsymbol{r}}_1 & \cdots & \tilde{\boldsymbol{r}}_{M-1} \end{bmatrix} = U\Sigma V^\top \tag{3.9}$$

Letting $U = \begin{bmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_L \end{bmatrix}$, An individual feature at a location $\boldsymbol{x}_m$ is computed according to

$$q_l(\boldsymbol{x}_m) = \boldsymbol{u}_l^\top \tilde{\boldsymbol{r}}_m, \qquad l = 1, 2, \ldots, L^*, \tag{3.10}$$

where L^* is the number of used features, which can be chosen based on the singular values

as described later. This notation is used to describe the lth feature, parameterized by a location vector. This is done to facilitate describing these features as a sample from one of L^* random fields, as will be done in Section 3.2. Examples of features from autocorrelations in this thesis are given in Figure 3.1.

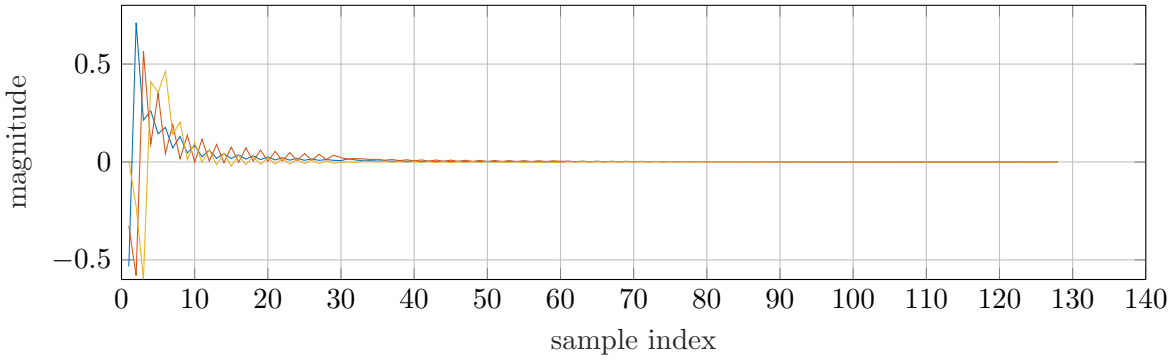


Figure 3.1: Example $\boldsymbol{u}$ vectors, which are the forms projected onto to get features. Thus these resemble the "shapes" of the features selected from the autocorrelation vectors.

These features can be grouped into a feature vector for a given reference location as

$$\boldsymbol{q}_m = \begin{bmatrix} q_0(\boldsymbol{x}_m) \\ q_1(\boldsymbol{x}_m) \\ \vdots \\ q_{L^*}(\boldsymbol{x}_m) \end{bmatrix}. \tag{3.11}$$

Where it is needed to convert a full signature vector to a vector of features, the matrix F can be used. One can construct F using the first L^* columns of U and taking the transpose

$$F = \begin{bmatrix} \boldsymbol{u}_0^\top \\ \boldsymbol{u}_1^\top \\ \vdots \\ \boldsymbol{u}_{L^*}^\top \end{bmatrix} \tag{3.12}$$

Using this definition, one can convert the matrix of reference measurements R into a matrix of reference features $Q = \begin{bmatrix} \boldsymbol{q}_0 & \boldsymbol{q}_1 & \cdots & \boldsymbol{q}_{M-1} \end{bmatrix}$ using

$$Q = FR. \tag{3.13}$$

To see why this works, and to motivate selection of the variable L^*, let $V = \begin{bmatrix} v_0 & v_1 & \cdots v_{L-1} \end{bmatrix}$, and note that $\Sigma = \begin{bmatrix} \sigma_0 & & & \vdots & \\ & \ddots & & 0 & \cdots \\ & & \sigma_{L-1} & \vdots & \end{bmatrix}$. Using these, a data vector $\tilde{\boldsymbol{r}}_i$ can be written as

$$R\boldsymbol{e}_i = \tilde{\boldsymbol{r}}_i = \sum_{l=0}^{L-1} \boldsymbol{u}_l(\sigma_l \boldsymbol{v}_l^\top \boldsymbol{e}_i) \tag{3.14}$$

with the elementary vector $\boldsymbol{e}_i$ (which has elements equal to zero except for the ith element equal to one). Using $\boldsymbol{e}_i$ in this manner extracts the ith column from the left multiplied matrix.

In this expression, the $\boldsymbol{v}_l^\top \boldsymbol{e}_i$ portion can be thought of how much of component l is contained in data vector i. Since $\boldsymbol{u}_l$, $\boldsymbol{v}_l$, and $\boldsymbol{e}_i$ are all unit vectors, this term will be between zero and one, with σ carrying the general magnitude information for that component. By construction of the singular value decomposition, the σ values are ordered from largest to smallest, making the $l = 0$ term to be the largest in magnitude overall, then $l = 1$ and so forth. The separated portion ($\sigma_l \boldsymbol{v}_l^\top \boldsymbol{e}_i$) for the lowest values of l can be considered the most significant components of the data vector r_i.

The $\boldsymbol{u}_l$ vector describes the "shape" of the particular component in the original data matrix, and when taken with the other components, allows reconstruction of the original data vectors. Using an incomplete selection of l terms still leads to an approximate reconstruction of the data matrix, and if later σ_l terms are small, the approximation can be quite accurate. In this research it is chosen to use the first L^* terms, such that σ_l is small for $l \geq L^*$ (relative to the earlier σ_l terms), effectively reducing the size of the data vectors without real loss of information. where L^* is selected such that for the singular values σ_n,

$\sum_{n=1}^{L^*} \sigma_n < 0.99\,\mathrm{Tr}(\Sigma)$. These $q_l(\boldsymbol{x}_m)$ are considered to be samples of a random field, $q_l(\boldsymbol{x})$.

The particular choice of F given in (3.12) is a simple means to extract the selected features. The $\boldsymbol{u}_l$ vectors are orthonormal, so $\boldsymbol{u}_i^\top \boldsymbol{u}_j = \delta_{ij}$, so applying $\boldsymbol{u}_j^\top$ to the summation in (3.14) will give the $\boldsymbol{u}_l$ term for $l = j$, while other terms will become zero. The rows of F thus extract the different feature components of the data matrix on different rows.

3.1.7 Singular vector component feature independence

Another advantage of this choice of features is that they are made to be uncorrelated, giving a level of approximate independence. To see this, consider the estimated covariance matrix from the original data.

$$\hat{\Sigma}_{\tilde{\boldsymbol{r}}} = \sum_{m=0}^{M-1} \tilde{\boldsymbol{r}}_m \tilde{\boldsymbol{r}}_m^\top = RR^\top \tag{3.15}$$

Now consider the corresponding covariance of the feature data matrix instead, and the (3.12),

$$\hat{\Sigma}_q = \sum_{m=0}^{M-1} q_m q_m^\top = QQ^\top = (FR)(FR)^\top \tag{3.16}$$

$$= FRR^\top F^\top \tag{3.17}$$

Then using (3.9),

$$FRR^\top F^\top = FU\Sigma V^\top V \Sigma^\top U^\top F^\top \tag{3.18}$$

and using the fact that $\boldsymbol{u}$ and $\boldsymbol{v}$ vectors are orthonormal sets, and that F is likewise a submatrix of U,

$$FU\Sigma V^\top V\Sigma^\top U^\top F^\top = FU\Sigma I\Sigma^\top U^\top F^\top \tag{3.19}$$

$$= \begin{bmatrix} I_l & \mathbf{0} \end{bmatrix} \Sigma\Sigma^\top \begin{bmatrix} I_{L^*} \\ \mathbf{0} \end{bmatrix} \tag{3.20}$$

$$= \Sigma^2_{0:L^*-1} \tag{3.21}$$

Where $\Sigma^2_{0:L^*-1} = \begin{bmatrix} \sigma_0^2 & & \\ & \ddots & \\ & & \sigma_{k-1}^2 \end{bmatrix}$. The sample covariance matrix for these components is diagonal, indicating uncorrelated components. If the components were in fact Gaussian in distribution, this would imply that the components were (at least approximately) independent, though this is of course not true in the general case.

3.2 Gaussian processes approach

A powerful approach for fingerprinting localization is to model the problem using Gaussian processes. In Section 2.3, a likelihood was obtained by the fitting a model to the L_2 distance to particular references measurements. While useful, this likelihood reduces the signature information to what is contained in these error values. It is preferable to use a model which more comprehensively uses all available signature infomation. This can be achieved by using a Gaussian random process model. Here, the signature is decomposed into "features" (as discussed previously in section 3.1.5) which are then modelled individually as independent Gaussian random processes (or equivalently as a form of vector Gaussian random process). The reference measurements taken during the offline training phase can be used to estimate feature components and model parameters. These parameters and measurements can then be used to produce a final distribution which will produce a likelihood for any location from a fingerprinting signature.

The Gaussian processes model, while more complicated, offers an impressive number of benefits, which are listed below.

1. The model automatically compensates for covariance between signature measurements taken in close proximity of each other.

2. The model naturally works with any distribution of reference measurements in three-dimensional space, not requiring an evenly sampled grid.

3. When coupled with an appropriate feature extraction method, GP automatically weights components of the fingerprint based on their spatial behavior, emphasizing the most useful information.

4. The model directly provides a likelihood for particular measurement positions, even when those positions are not found in the reference measurements

5. After an initial processing step, GP can be quite computationally efficient for localization.

3.2.1 Gaussian Processes Model

In this fingerprinting method, autocorrelation or channel features are modeled over the localization region as set of independent Gaussian random processes.A Gaussian process is a random processes for which every finite set of samples of the process are jointly Gaussian distributed (as explained in Rasmussen and Williams [32]).

Let the features $q_l(\boldsymbol{x})$ be samples from a Gaussian random process Q_l. From this, each $q_l(\boldsymbol{x})$ will be a sample from a Gaussian random variable $Q_l(\boldsymbol{x})$. The set of reference features will then be samples from the random variables of the form $Q_l(\boldsymbol{x}_m)$, which will be jointly Gaussian or

$$\begin{bmatrix} Q_l(\boldsymbol{x}_0) \\ Q_l(\boldsymbol{x}_1) \\ \vdots \\ Q_l(\boldsymbol{x}_{L-1}) \end{bmatrix} \sim \mathcal{N}(\mu_{\mathbf{l}}, \boldsymbol{\Sigma}_{\mathbf{l}}), \tag{3.22}$$

for some mean μ and covariance Σ_l for each feature l. Some further constraints are then applied to make the model fitting process more tractable.

The Gaussian random processes are restricted to have the following attributes.

1. Features share an underlying mean across the region, or $E[Q_l(\boldsymbol{x})] = E[Q_l(\boldsymbol{y})]\ \forall\ \boldsymbol{x}, \boldsymbol{y}$ in region of interest.

2. Features are independent, that is $Q_i(\boldsymbol{x}) \perp\!\!\!\perp Q_j(\boldsymbol{y})$ for all $i \neq j$ and any $\boldsymbol{x}$ and $\boldsymbol{y}$. Though some degree of interdependent features can be handled, this assumption makes estimation and processing much simpler. Appropriate feature selection/extraction can help this assumption to at least approximately hold.

3. Covariance between a feature at two different positions depends only on a simple function of the distance between them, that is $\text{Cov}(Q_l(\boldsymbol{x}), Q_l(\boldsymbol{y})) = f(\|\boldsymbol{x} - \boldsymbol{y}\|)$. More specifically, the Gaussian kernel function is used, which is $\gamma_l \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2d_l^2}\right)$.

4. Samples of features have some additional independent measurement noise with variance σ_l^2.

With these in place the distribution from (3.22) becomes

$$\begin{bmatrix} Q_l(\boldsymbol{x}_0) \\ Q_l(\boldsymbol{x}_1) \\ \vdots \\ Q_l(\boldsymbol{x}_{L-1}) \end{bmatrix} \sim \mathcal{N}(\mu_l \mathbf{1}, \sigma_l^2 I + \gamma_l K_l), \tag{3.23}$$

where $[K_l]_{i,j} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|}{2d_l^2}\right)$.

The parameters σ_l^2, γ_l, and d_l characterize the Gaussian processes behavior on feature l. These parameters can be estimated from the reference data.

3.2.2 Gaussian process model parameter estimation

In order to use the Gaussian process model for localization, it is necessary to find suitable model parameters using the available reference data. As the features are assumed to be independent, the model parameters are individual to each feature, and will each be estimated independently. That is, for each feature l, the parameters μ_l, σ_l^2, γ_l, and d_l will be found by solving an optimization based on the measurements of that feature extracted from the original reference measurements.

For brevity, the collection of features obtained from the full set of reference measurements is denoted as $\mathcal{Q}$, so $\mathcal{Q} = \{q_l(\boldsymbol{x}_m) \mid l = 0 \dots L^* - 1,\ m = 0 \dots M - 1\}$.

3.2.3 Maximum likelihood parameter estimation

As explained by Ferris et al. in [25], Gaussian process model parameters can be estimated using a maximum likelihood approach.

In this approach the parameters are found as the solution to the optimization problem

$$\max_{\mu_l.\sigma_l^2,\gamma_l,d_l} ((2\pi)^{-\frac{d}{2}}|\Sigma_l{}^{-1}|) \exp\left[-\frac{1}{2}(\boldsymbol{q} - \mu_l\mathbf{1})^\top \Sigma_l^{-1}(\boldsymbol{q} - \mu_l\mathbf{1})\right], \tag{3.24}$$

where

- μ_l is the mean of the feature over the relevant space
- σ_l^2 is the noise variance associated with an individual feature measurement
- γ_l is the variance associated which is shared locally over physical distance
- d_l is a scale parameter describing the distance over which distance related covariance varies.
- $\boldsymbol{q}_l = \begin{bmatrix} q_l(\boldsymbol{x}_0) & \cdots & q_l(\boldsymbol{x}_{m-1}) \end{bmatrix}^\top$ is a vector containing measured features

- $\Sigma_l = \sigma_l^2 I + \gamma_l K_l$ is the covariance matrix, which also contains distance information.

It is likewise recommended by Ferris et al. that a conjugate gradient descent method be used to solve this optimization problem. In this research a simple gradient descent solver was implemented for this optimization problem to investigate the feasibilty of solving this problem. While this method seems to yield accurate results, it is fairly time consuming to run and did not ultimately scale well enough to use for much of this work. The high computational cost comes primarily from the difficulty of performing the large matrix operations involving the inverse of the covariance matrix, which are necessary to calculate gradients for the descent method. The difficulties in solving this problem efficiently are the subject or other research, such as by Ambikasaran et al. [33] and Das et al. [34].

Still, where computational resources allow, solving the maximum-likelihood problem is likely to produce the best model parameters, and as this step is only needed in the offline training phase, such an approach may still be appropriate in many systems. Due to these difficulties, and the need to solve this problem a large number of times for testing purposes, an alternative least-squares approach was developed.

3.2.4 Modified least-squares parameter estimation

As an alternative to the maximum-likelihood approach, the following constrained least-squares approach was developed for use in this research, due to its significantly lower computational complexity. This method chooses parameters which minimize the difference between the model-produced covariance matrix, and a covariance estimate obtained from the data.

In this approach, instead of maximizing the likelihood function directly, the parameters are optimized against a less constrained maximum-likelihood estimate.

First, an approximation for the mean of a particular layer is used as

$$\hat{\mu}_l = \frac{1}{M} \sum_{m=0}^{M-1} q_l(\boldsymbol{x}_{m-1}) \tag{3.25}$$

This is equal to the maximum-likelihood estimate for a mean when the covariance matrix is a multiple of an identity matrix (or the variables are indepentent and identically distributed). The covariance is not an identity for this model, but in general will not lead to a significantly different mean estimate than the more correct estimate.

The proper maximum-likelihood estimate for the mean is given by

$$\hat{\mu}_l = \frac{\mathbf{1}^\top \Sigma_l^{-1} \boldsymbol{q}}{\mathbf{1}^\top \Sigma_l^{-1} \mathbf{1}} = \frac{\sum_{m=0}^{M-1} [\Sigma_l^{-1} \boldsymbol{q}]_m}{\sum_{i=0}^{M-1} \sum_{j=0}^{M-1} [\Sigma_l^{-1}]_{ij}} \tag{3.26}$$

However, this requires the use of the covariance matrix, which has not yet been estimated. This can still be used in an iterative method, by alternating estimating the mean using a covariance estimate and the covariance using this estimated mean, but this was not performed in this research.

Using $\boldsymbol{q}_l = \begin{bmatrix} q_l(\boldsymbol{x}_0) & \cdots & q_l(\boldsymbol{x}_{m-1}) \end{bmatrix}^\top$ as before, the maximum likelihood for estimation of the covariance of a general Gaussian random vector set is simply the rank one matrix given by

$$\hat{\Sigma}_{ML} = (\boldsymbol{q}_l - \hat{\mu}_l \mathbf{1})(\boldsymbol{q}_l - \hat{\mu}_l \mathbf{1})^\top. \tag{3.27}$$

The objective function then fits the parameters against this in a least squares sense

$$\begin{aligned} \min_{\sigma_l^2, \gamma_l, d_l} \quad & \|\hat{\Sigma}_{ML} - (K_l + \sigma_l^2 I)\|^2 \\ \text{subject to} \quad & \sigma_l^2 > 0 \\ & \gamma_l > 0 \\ & d_l > 0. \end{aligned} \tag{3.28}$$

Using $\boldsymbol{s}_{ML} = \text{vec}(\hat{\Sigma_{ML}})$, $\boldsymbol{\delta} = \text{vec}(I)$, and $\boldsymbol{\kappa}(d_l) = \text{vec}(K_l)$, this can be expressed as the vectorized problem

$$\begin{aligned} \min_{\sigma_l^2, \gamma_l, d_l} \quad & \|\boldsymbol{s}_{ML} - \begin{bmatrix} \boldsymbol{\delta} & \boldsymbol{\kappa}(d_l) \end{bmatrix} \begin{bmatrix} \sigma_l^2 \\ \gamma_l \end{bmatrix} \|^2 \\ \text{subject to} \quad & \sigma_l^2 > 0 \\ & \gamma_l > 0 \\ & d_l > 0. \end{aligned} \tag{3.29}$$

If d_l is known and the positivity constraints on σ_l^2 and γ_l are ignored, then the problem becomes linear in σ_l^2 and γ_l and can be solved directly by

$$\begin{bmatrix} \sigma_l^2 \\ \gamma_l \end{bmatrix} = \begin{bmatrix} M^2 & \boldsymbol{\delta}^\top \boldsymbol{\kappa} \\ \boldsymbol{\delta}^\top \boldsymbol{\kappa} & \boldsymbol{\kappa}^\top \boldsymbol{\kappa} \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{\delta}^\top \boldsymbol{s}_{ML} \\ \boldsymbol{\kappa}^\top \boldsymbol{s}_{ML} \end{bmatrix}. \tag{3.30}$$

Here, if a negative value is given for σ_l^2 or γ_l, then that variable can be set to zero, and the other one can be found using one of

$$\gamma_l = \frac{\boldsymbol{\kappa}^\top \boldsymbol{s}_{ML}}{\boldsymbol{\kappa}^\top \boldsymbol{\kappa}} \qquad \text{or} \qquad \sigma_l^2 = \frac{\boldsymbol{\delta}^\top \boldsymbol{s}_{ML}}{M^2}. \tag{3.31}$$

Since d_l is not known, one can select several feasible values for d_l and identify the corresponding optimal σ_l^2, γ_l, and error values associated with each value. While this method is not likely to produce model parameters which are as well fit to the data as the maximum-likelihood parameter estimation method, this method still seemed to produce reasonably suitable model parameters, which performed adequately for localization.

Sensible limits for the range of this variable can be drawn from prior knowledge. The variable d_l sets a scale at one location's measurement is similar to another location's measurement in the environment. To set bounds for this, it seems reasonable that if d_l was so small that only 1% of the kernel term remained at the distance between measurements, it is not useful for localization because it would be very small unless a transmission occured

right where a measurement was taken. Likewise, if d_l was so large that 99% of the kernel term remained at the distance between measurements, then the feature likely varied so little that it would not be useful in distinguishing between locations in the environment. This reasoning gives the d_l bounds

$$d_{l,\text{bound}} = \sqrt{\frac{d_{\text{ref}}}{-2\ln(p_{\text{critical}})}} \tag{3.32}$$

For example, if $d_{\text{ref}} = 1\text{m}$, and p_{critical} was 0.01 and 0.99 for the lower and upper bound respectively, the corresponding $d_{l,\text{bound}}$ values would be 0.3295m and 7.0533m.

Potential values for d_l in such a range can then be evaluated according the the objective function in (3.29), giving a plot, such as that shown in Figure 3.2. As illustrated there, it

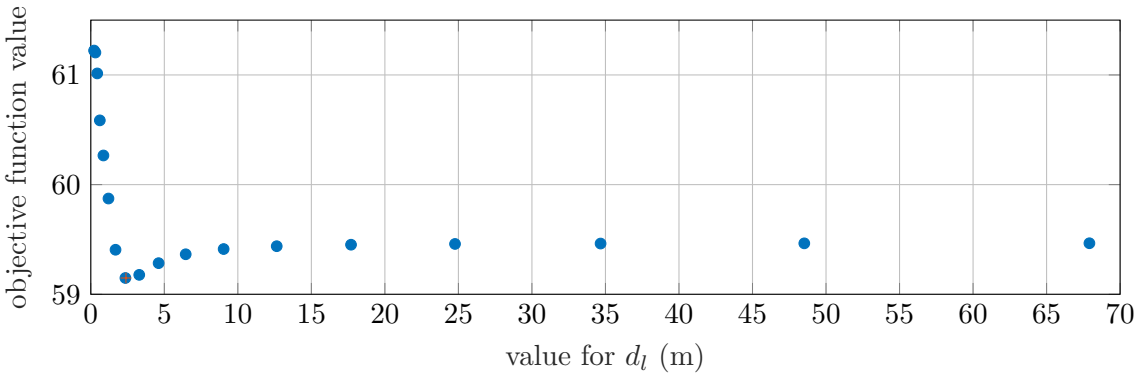


Figure 3.2: The objective function from (3.29) evaluated with different values for d_l (using the computed optimal values for other model parameters)

is also possible to interpolate to find a better minimum point without further evaluations. Even as few as 5 to 15 evaluations gave fairly effective results. Specifically evaluation was done in this work using 6 possible d_l values and p_{critical} values of 0.001 and 0.999. Rather than using linear spacing for these points, it is better to use logarithmic or other spacing, as the objective function often has more variation at the lower end. This improves optimality of the d_l selection with fewer evaluated points.

It is also worth noting that if γ_l is zero, then this suggests that a feature does not

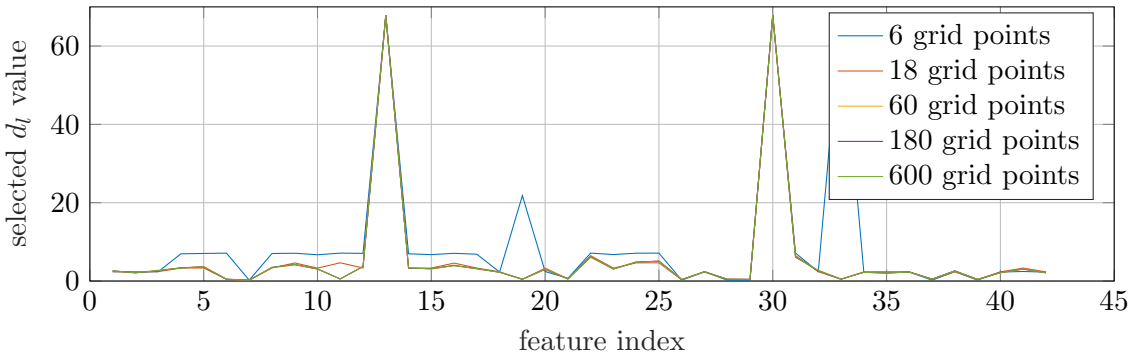


Figure 3.3: Selected d_l values for all features when different numbers of d_l values are tested. Larger values produce values closer to the optimum, but parameters change relatively little past using 18 points.

have shared information between nearby measurements, and is not likely to be useful for localization. One can manually drop this feature from final likelihood equations, though fortunately the resulting zeros in the relevant covariance matrices will lead that feature to have no impact on the final likelihood estimate (though it will still waste computational resources).

3.2.5 Final adjustments

With either method, the optimum result can give parameters which are not ideal for estimation. Particularly an exceptionally small value of σ_l^2 can cause computational problems, and also can give excessive confidence in measurements for that feature. To avoid these issues, σ_l^2 is increased to $0.1\gamma_l$ if it is below that value, that is

$$\sigma_{l,\text{adjusted}} = \begin{cases} \sigma_l, & \sigma_l \geq 0.1\gamma_l \\ 0.1\gamma_l, & \text{otherwise} \end{cases}. \tag{3.33}$$

This avoids putting excessive confidence in individual measurements, which may also help to compensate for slight environmental changes after reference measurements are collected or other factors limiting the reliability of individual measurements.

3.2.6 Localization using the Gaussian random process model

The ultimate goal of the model is to obtain an expression for the likelihood of a transmitter being in a given location, based on its signature and the available reference measurements.

After features are obtained from a received signal of interest (by computing autocorrelation, performing preprocessing steps and performing feature extraction), the likelihood associated with that signal originating form an arbitrary location is desired. That is, for some vector of features from a received signal $\boldsymbol{q}$, what is the likelihood that $\boldsymbol{q}$ was produced from a transmitter located at $\boldsymbol{z}_k$, and using reference information $\mathcal{Q}$. For each individual feature, this can be written as a pdf on $Q_l(\boldsymbol{z}_k)$ conditioned on the available reference information $\mathcal{Q}$, given as $f_{Q_l(\boldsymbol{z}_k)|\mathcal{Q}}(q_l)$. Leveraging the fact that the features are independent, one can find the overall probability using all of the features as the product of the probabilities of each individual feature, as in

$$f_{\boldsymbol{Q}(\boldsymbol{z}_k)|\mathcal{Q}}(q) = \prod_{l=1}^{L^*} f_{Q_l(\boldsymbol{z}_k)|\mathcal{Q}}(q_l). \tag{3.34}$$

To obtain the distribution on individual features at different locations, one can use the fact that these features are modelled as Gaussian. One can use the property that conditional Gaussian random variables are also Gaussian with mean and standard deviation given by

$$\mu_{X|Y=y} = \mu_X + \Sigma_{XY}\Sigma_{YY}^{-1}(y - \mu_Y) \tag{3.35}$$

$$\Sigma_{X|Y=y} = \Sigma_{XX} - \Sigma_{XY}\Sigma_{YY}^{-1}\Sigma_{YX}. \tag{3.36}$$

The model described in Gaussian model given in (3.23) is used with these equations to find the conditional mean and covariance, by conditioning on the available reference measurements. From this model the conditional features are Gaussian distributed for any particular candidate location $\boldsymbol{z}_k$ as $Q_l(\boldsymbol{z})|\mathcal{Q} \sim \mathcal{N}(\mu_{Q_l(\boldsymbol{z}_k)|\mathcal{Q}}, \Sigma_{Q_l(\boldsymbol{z}_k)|\mathcal{Q}})$, with

$$\mu_l(\boldsymbol{z}) = \mu_l + \boldsymbol{k}(\boldsymbol{z})^\top \Sigma_l^{-1}(\boldsymbol{q}_l - \mu_l \mathbf{1}) \tag{3.37}$$

$$\sigma_l^2(\boldsymbol{z}) = \sigma_l^2 - \boldsymbol{k}(\boldsymbol{z})^\top \Sigma_l^{-1} \boldsymbol{k}(\boldsymbol{z}) \tag{3.38}$$

where

$$\boldsymbol{k}(\boldsymbol{z}) = \begin{bmatrix} \exp\left(-\frac{\|\boldsymbol{z}-\boldsymbol{x}_0\|}{2d_l^2}\right) \\ \vdots \\ \exp\left(-\frac{\|\boldsymbol{z}-\boldsymbol{x}_{M-1}\|}{2d_l^2}\right) \end{bmatrix} \tag{3.39}$$

and $\Sigma_l = \sigma_l^2 I + \gamma_l K_l$ as determined by the estimated model parameters on the reference measurements.

This conditional mean and variance can be calculated for an array of positions $\boldsymbol{z}_k$ in the region of interest for each layer l. The mean and variance for one such feature is depicted in Figure 3.4, and summarizes the posterior distribution on this feature for all candidate locations in the region of interest. Note that variance increases in regions with fewer measurements, as there is less information constraining these points. This can also be viewed as an array of Gaussian random variables, with a mean and variance for each point. When localization is performed, a likelihood can be calculated for each of these random variables. Depictions of additional means and features are provided and discussed in Section 4.1.2.

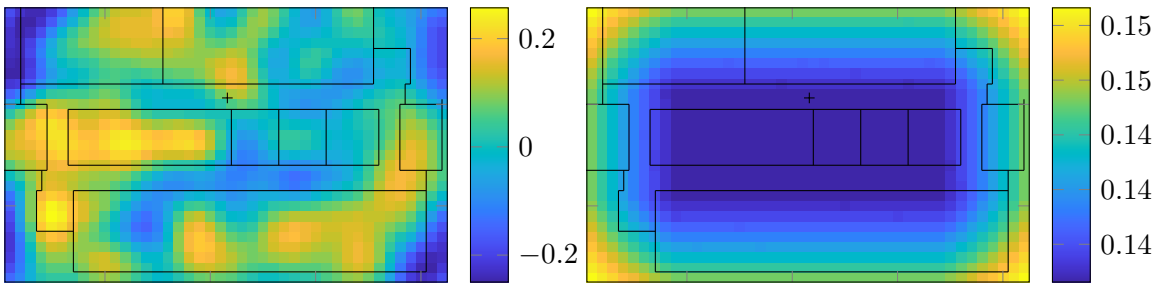


Figure 3.4: (Left) The mean for the Gaussian process fit to a single feature, and (Right) the variance for the same feature.

With the mean and variance calculated, the distribution for a particular feature at an arbitrary candidate point is known. A final likelihood can be calculated for each candidate position $\boldsymbol{z}_k$ by using the Gaussian pdf:

$$f_{\boldsymbol{Q}(\boldsymbol{z}_k)|\mathcal{Q}}(\boldsymbol{q}) = \prod_{l=1}^{L^*} f_{Q_l(\boldsymbol{z}_k)|\mathcal{Q}}(q_l) = \prod_{l=1}^{L^*} (\sqrt{2\pi\sigma_l^2(\boldsymbol{z})})^{-1} \exp\left[-\frac{1}{2\sigma_l^2(\boldsymbol{z})}\|q_l - \mu_l(\boldsymbol{z})\|^2\right], \tag{3.40}$$

where $\mu_l(\boldsymbol{z})$ and $\sigma_l^2(\boldsymbol{z})$ are the conditional mean and variance given in (3.37) and (3.38). In practice it is best to instead calculate the log of the terms of these equations and take their sum, rather than product, and finally take the exponential of the final result to avoid issues with limitations of numerical floating point operations.

Using (3.40), a likelihood for a transmitted signal-of-interest can be calculated for each candidate location $\boldsymbol{z}_k$. As discussed in Section 2.3, the position with the highest likelihood becomes the estimated transmitter position.

CHAPTER 4

Considerations and Performance of Autocorrelation-Based Fingerprinting

The results in this section are essentially the same as those previously published by Ipson and Moon [28], though some minor code adjustments and slightly different parameters have produced slightly different results.

4.1 Relationship between autocorrelation and position changes

While the full relationship of position to autocorrelation is not tractable, analysis of simple case with one reflector can provide useful insights into this relationship.

Received signal autocorrelation functions stem from the underlying structure of path delays and amplitudes produced by the propagation environment. In general, slight changes in emitter position will give slight changes in the lengths of these paths, leading to slight changes in the delays and amplitudes leading to a smooth change in the autocorrelation function. If a small motion results in a path being obstructed, that can lead to more abrupt autocorrelation changes, which does cause a negative but tolerable effect on signature smoothness. The scale of these changes is influenced by the emitter bandwidth, which is discussed some in Section 4.1.3.

4.1.1 Spatial behavior of channel in single mirror environment

As previously noted, it is not practical to fully model how the channel varies with small changes in position in a complex environment, one can gain important intuition by investigating a simplified situation. Consider the scenario of a single transmitter and a single receiver with one infinite reflective wall. In a real reflector, reflectivity changes based on surface angle, however this effect is ignored in the following results to maintain simplicity. In the discussion here, the receiver position is fixed and the transmitter position effect is varied, though the analysis is identical if these are reversed.

The most important factor defining the channel properties in this environment is the difference between the direct path-length and the reflected path-length. The path-length difference corresponds to a time delay and phase difference which may be identifiable at the receiver. Path-length also plays a role in the power of the received signal, though this is less sensitive to slight changes than the path-length difference. The direct path length and reflected path length are given by

$$d_{\text{direct}} = \sqrt{x^2 + (y_1 - y_2)^2} \tag{4.1}$$

$$d_{\text{reflected}} = \sqrt{x^2 + (y_1 + y_2)^2}. \tag{4.2}$$

Using this relationship, one can view the difference in path length with respect to the relative position of the transmitter/receiver. This is shown in Figure 4.1 (top left).

Observe from Figure 4.1 (Top left) that the path-length-difference ranges from zero to twice the distance between the receiver and the wall for a single reflection, functionally limiting the possible relative time-delay. Paths with multiple reflections could of course still be larger, but will also be weaker. It is also apparent that the path-length-distance is constant along lines extending from the transmitter. This is also visible in Figure 4.1 (Top right), where the relative phase is depicted for the wavelength $\lambda = 1\text{m}$ (or $f_c = 300\text{MHz}$). Note that the phase is constant alongs these lines, and changes at different rates in the tangential direction. It can be shown from (4.1) and (4.2) that the width of these bands ranges from approximately from $\frac{x\lambda}{2y}$ along the wall to $\frac{\lambda\sqrt{y_2^2 - y_1^2}}{2y_1}$ far from the wall, and $\frac{\lambda}{2}$ when y_2 is smaller than y_1 and x is small.

In Figure 4.1 (Bottom left), the power of sinusoid (again with $\lambda = 1\text{m}$ or $f = 300\text{MHz}$) is depicted with a constant reflection coefficient of $R = 0.8$. This shows how the phase difference in Figure 4.1 (Top right) will apply when the time-difference of these paths is small relative to the inverse of the signal bandwidth. Overall change may be slower since net phase effect, shown in Figure 4.1, doesn't change as quickly as the relative phase of the two paths.

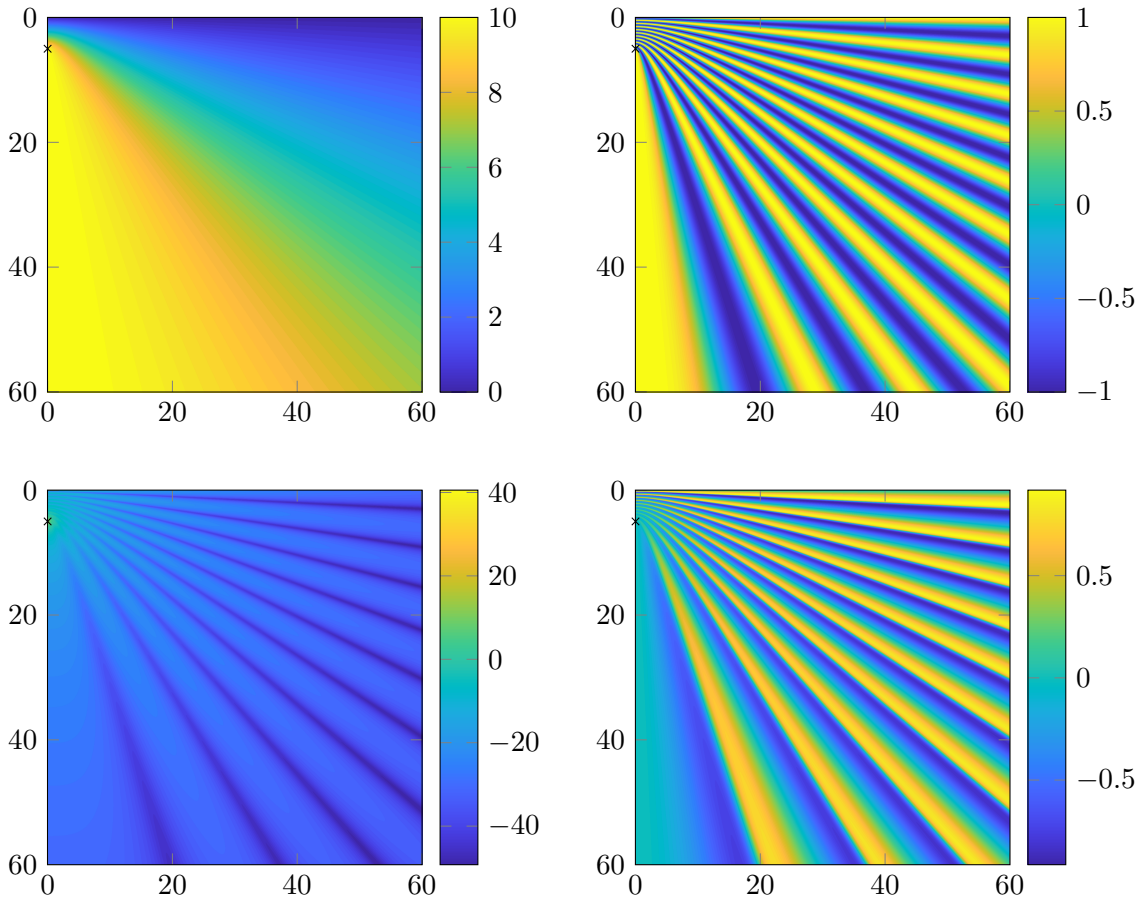


Figure 4.1: (Top left) Path-length difference for receiver at different transmitter positions, (Top right) phase-difference at different transmitter positions for wavelength of $\lambda = 1$m, (Bottom left) approximate amplitude of a sinusoid with wavelength of $\lambda = 1$m at different transmitter positions. (Bottom right) phase change due at receiver due to reflected path

The phase and time-difference changes are ultimately what determine the ultimate effect of the channel which is observable in the autocorrelation. For a particular reflector, like that depicted here, that does not change at all along the lines radiating from the source in Figure 4.1. In this direction, the channel does not change as much due to that particular component.

The combination of multiple reflectors, reflections, and occlusion may lead to much more complicated phenomena, however it seems likely that channel behavior will still show elements of this behavior. When a path becomes obstructed by an obstacle, it may cause a more substantial change which is more abrupt. Still, this provides at least a starting idea of

the scale over which the channel is likely to vary significantly. Encouragingly, this suggests that even with smaller wavelengths, there is likely to be some stability over physical space in some regions. Unfortunately this still suggests that in some regions will have variation over a distance scale of half a wavelength, which would require a map test point every quarter wavelength (to reach nyquist spacially), a daunting task for even a 1m wavelength signal!

4.1.2 Spatial behavior of autocorrelation features from simulation, and Gaussian process model

The Gaussian process method described in Chapter 3 produces features associated with each reference measurement location. These are depicted graphically in Figures 4.2 — 4.5, along with the Gaussian process mean produced for that feature. The Gaussian process mean offers some insight into regional trends produced by the more individually varied features.

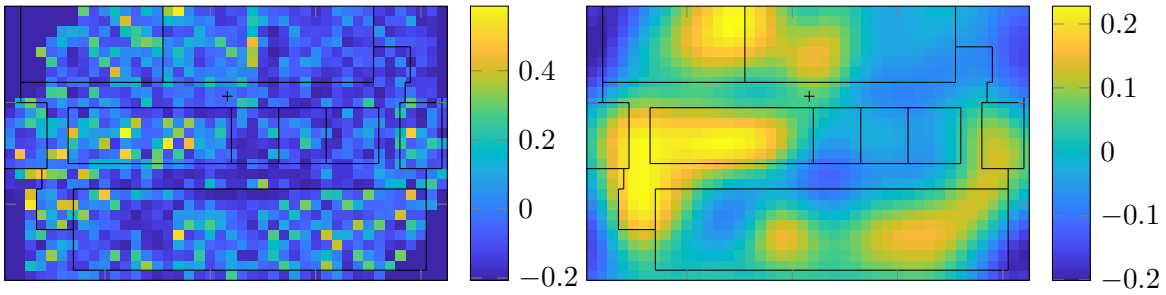


Figure 4.2: (Left) First feature derived from reference measurements over region at a lower bandwidth of 32 MHz. (Right) the Gaussian process model means derived from these features.

4.1.3 Relationship between autocorrelation and bandwidth

The identifiable time-of-arrival information is closely related to the bandwidth of a signal. This can be mathematically understood simply by the established Fourier transform property:

$$\mathcal{F}^{-1}\{H(f)\} = h(t) \implies \mathcal{F}^{-1}\{H(f/w)\} = h(w \cdot t), \tag{4.3}$$

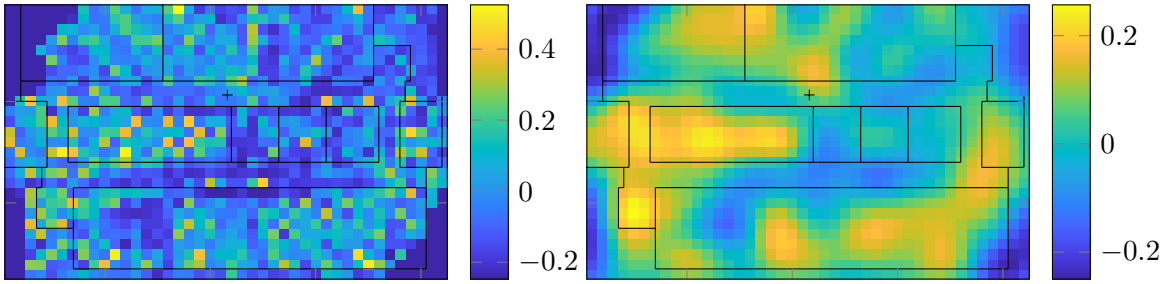


Figure 4.3: (Left) First feature derived from reference measurements over region at a bandwidth of 64 MHz. (Right) the Gaussian process model means derived from these features.

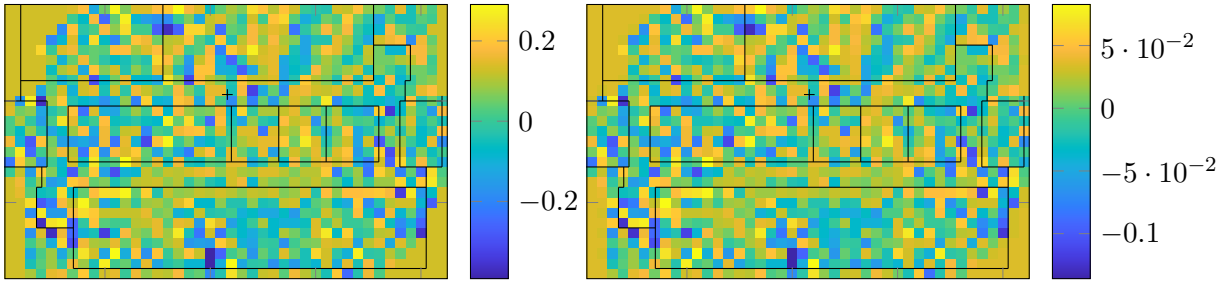


Figure 4.4: (Left) Second feature derived from reference measurements over region at a bandwidth of 64 MHz. (Right) the Gaussian process model means derived from these features. Note that in this case the chosen Gaussian process model does not have significant spatial correlation; this feature will not significantly contribute to localization.

where $h(t)$ is some signal as a function of time t (or equivalently a channel impulse response), $\mathcal{F}$ is the continuous-time Fourier transform, and w is some scaling constant. From this relationship, it can be seen that if a signal is "widened" in the frequency domain by increasing w (thus increasing the signal bandwidth), its corresponding form in the time-domain becomes more narrow. Very abrupt or high-speed changes make identifying the corresponding time of this change easier, which will occur for high values of w, corresponding to a widened version of $H(F)$.

To see how this effect manifests in autocorrelation functions, consider a basebanded signal which is bandlimited to a bandwidth of B. This is equivalent to passing the signal through an ideal lowpass filter, which is equivalent to convolution with $\text{sinc}(\frac{t}{B})$. Note that this function broadens as B decreases. The effect of bandlimiting a signal correspondingly affects the autocorrelation, which is tied to the transfer function. If the signal is bandlim-

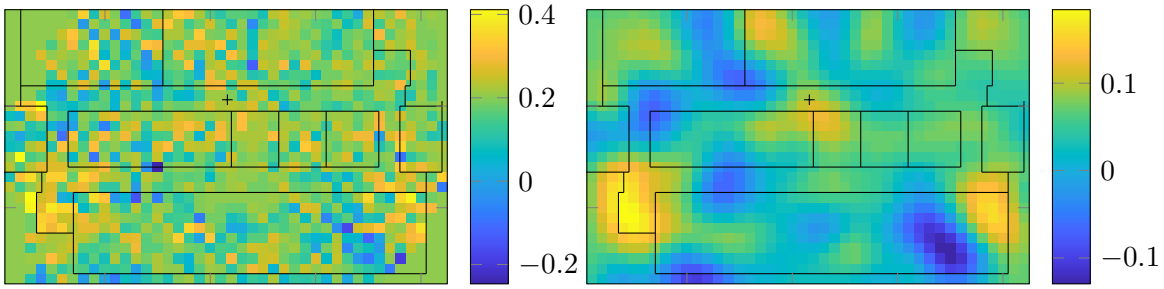


Figure 4.5: (Left) Third feature derived from reference measurements over region at a bandwidth of 64 MHz. (Right) the Gaussian process model means derived from these features.

ited, the apparent transfer function will likewise be limited. If one considers in the frequency domain a received signal $Y(f)$, transfer function $H(f)$, and original signal $X(f)$, then it is given that $Y(f) = H(f)X(f)$. As such, $X(f) = 0$ implies $Y(f) = 0$, and nothing can be inferred about $H(f)$ from $Y(f)$. Any Identified information about $H(f)$ will thus be bandlimited as well. The autocorrelation path information will likewise obscured as bandwidth decreases, as it is derived from the transfer function information. This is illustrated in Figure 4.6

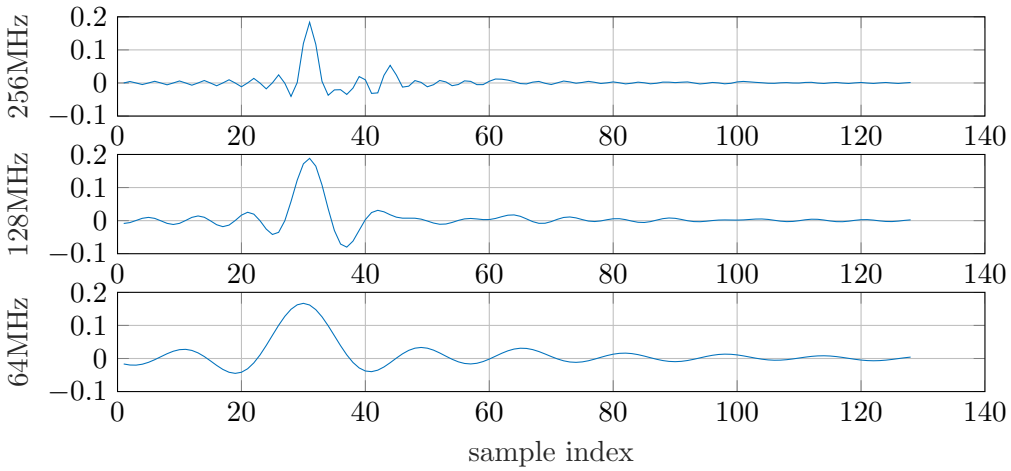


Figure 4.6: Channel impulse response at 256 MHz, 128 MHz, and 64 MHz bandwidth

Finally, a Cramer-Rao lower bound on the time-delay estimation variance is given by Wang et al. [35] as

$$\sigma_\tau^2 = \frac{1}{SNR \cdot \beta_0^2} \tag{4.4}$$

where β_0^2 is the mean square bandwidth, given by

$$\beta_0^2 = \frac{(2\pi)^2 \int_{-\infty}^{+\infty} f^2 |s(f)|^2 df}{\int_{-\infty}^{+\infty} |s(f)|^2 df} \tag{4.5}$$

Mean square bandwidth is a form of bandwidth measure, and it can be seen here that time-delay estimation variance increases as bandwidth decreases, further establishing the importance of bandwidth in estimating time-relevant features.

4.1.4 Coherence bandwidth

As discussed, the bandwidth plays a significant role in the degree to which timing resolution is possible. This timing resolution in turn translates to resolution of path-lengths in the propagation environment. However, not all environments are created equally, and differences between direct-path lengths and indirect-path lengths are dependent on physical distances between reflective features in the environment. While full analysis of the effects of different environments is beyond the scope of this research, it is worth one metric which can be used to compare different environments.

Coherence bandwidth is a measure describing the scale over which the frequency response in an environment can be considered "flat". As given in Pahlavan and Levesque [36], it is calculated from the RMS delay spread, which is given for a particular channel by

$$\tau_{\text{rms}} = \sqrt{\bar{\tau^2} - (\bar{\tau^1})^2}, \tag{4.6}$$

where

$$\bar{\tau^n} = \frac{\sum_i \tau_i^n |\alpha_i|^2}{\sum_i |\alpha_i|^2}, \tag{4.7}$$

with complex path gains α_i and path delays τ_i relative to the shortest delay (so $\tau_0 = 0$) for a particular channel.

The coherence bandwidth can then be calculated for a particular path as

$$\beta_{\text{coh}} = \frac{1}{\tau_{\text{rms}}}. \tag{4.8}$$

Pahlavan and Levesque also remark that some sources use the mean delay spread $\bar{\tau}^1$ instead of the RMS delay spread for calculating coherence bandwidth, but the RMS version is used here.

The coherence bandwidth for a particular receiver location is depicted in Figure 4.7.

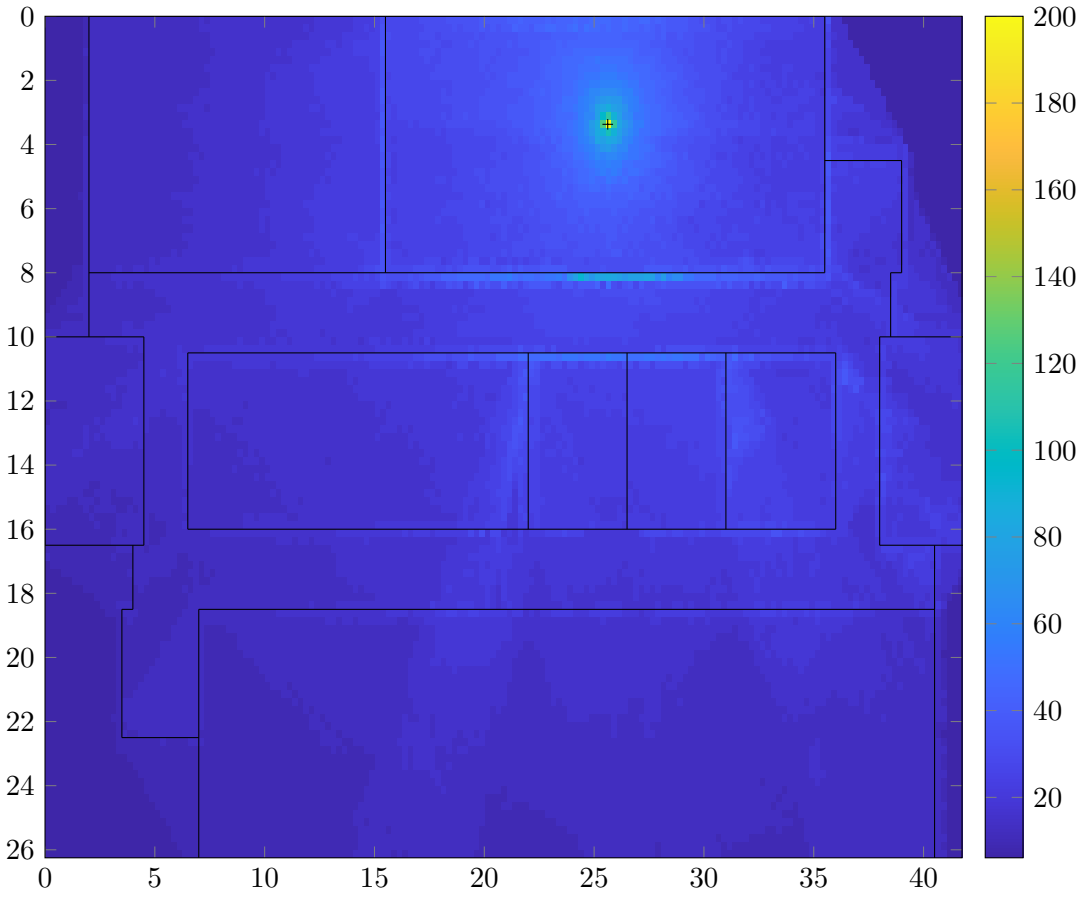


Figure 4.7: Root-mean-square coherence bandwidth associated with the transfer function between the indicated point and a grid of points throughout the environment. Bandwidth is given in MHz.

If the average value is taken for all points depicted in Figure 4.7, a value of 17.058 MHz is obtained. This does not take into account all possible receiver locations for the given environment, but is likely still useful for describing the test environment used in this thesis.

4.2 Evaluation of performance

To evaluate the performance of this localization approach, tests were performed using channels generated using the ray-tracing methods discussed in Chapter 6. A variety of comparisons and tests at different bandwidths were performed, the methods used and final results are presented in this section.

4.2.1 Simulation setup

The ray-tracing simulation was run on a simplified model of a lab environment, with no furniture or other smaller features modelled. Positions were selected for simulated receivers along with a grid of transmitter points across the area, spaced every half meter. A subgrid of transmitters with one-meter spacing was assigned for use as reference measurements for training offline localization, with a subset of the remaining points used as test points for evaluating localization performance. While the Gaussian processes approach can use an arbitrary candidate grid which is independent from the actual positions where reference measurements are taken, the candidate position grid here is chosen to be the same positions as the reference measurements. This is done so that it can use the same candidate grid as the L_2 one-nearest-neighbor approach it is being compared to, allowing for a more fair comparison. The simulation environment and receiver and transmitter positions are shown in Figure 4.8, where the blue points represent reference positions $\{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_{M-1}\}$, the red points are used as test locations $\{\boldsymbol{z}_0, \ldots, \boldsymbol{z}_{T-1}\}$, and the black circles indicate receiver locations.

Reflection and transmission coefficients were used base on Table 5.3 using metallicized glass, and are depicted in Figure 4.9.

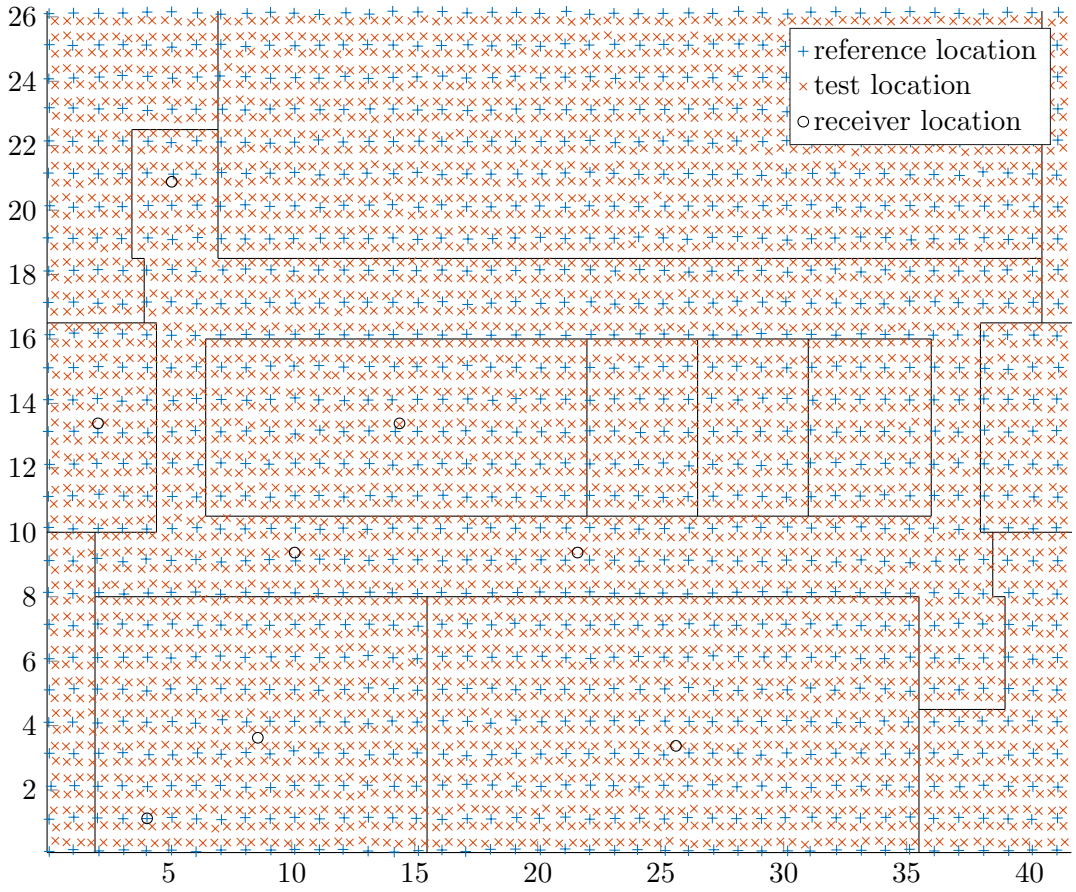
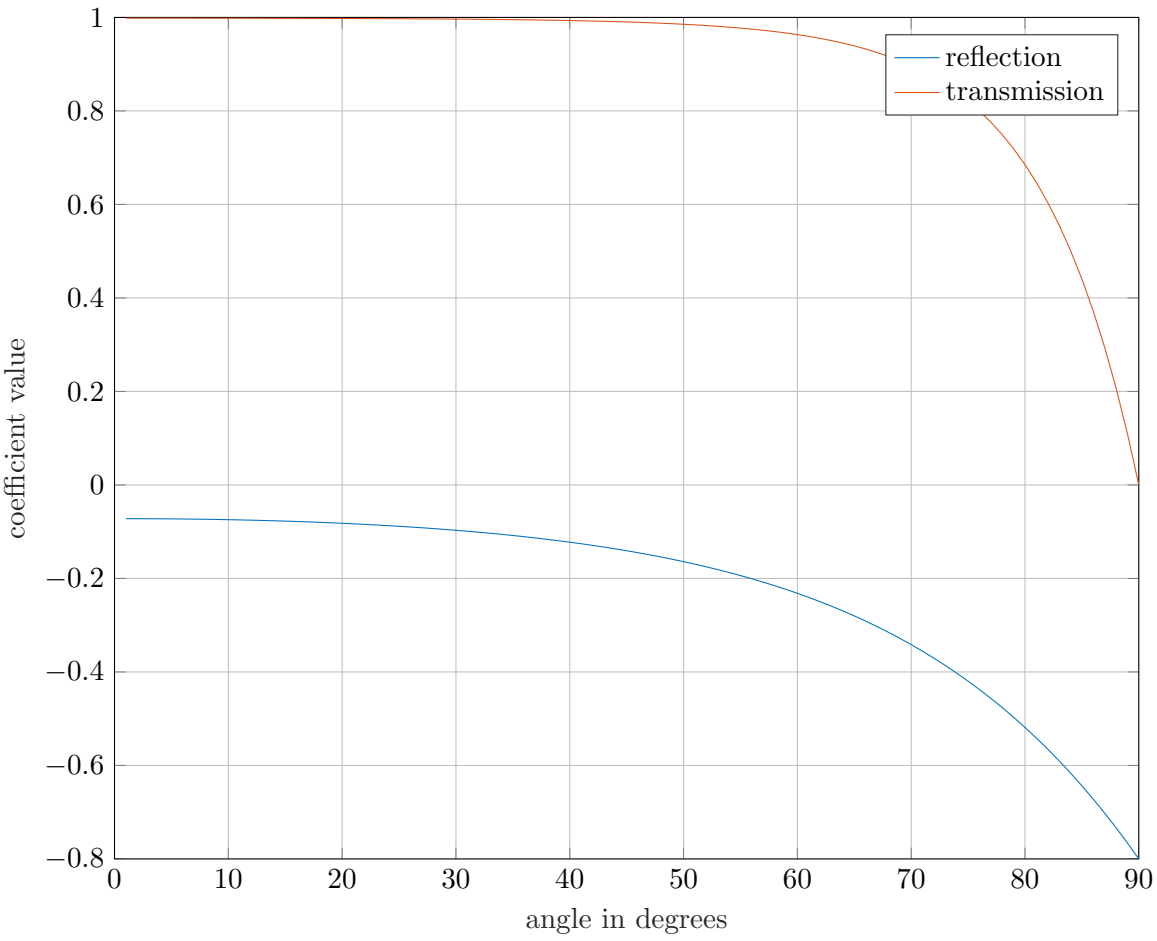


Figure 4.8: Map of simulation test environment with receiver and transmitter locations



width

Figure 4.9: Reflection and transmission coefficients used for wall interactions in simulation, corresponding to a slab of metallicized glass using values from Table 5.3

4.2.2 Evaluation method

After generating the channel impulse responses for each receiver-transmitter pair, the channels were used to test a variety of scenarios and variations. The autocorrelation was calculated using these channel impulse responses and fingerprinting was performed using the assigned reference measurements and testing with each test point. Since the autocorrelation used is calculated directly from the channel impulse response in this manner, it is equivalent to using a band-limited white noise signal for both the reference measurements and test measurements. Tests were performed with each receiver independently, with results for each single-receiver system depicted as a separate data point in Figures 4.10 — 4.16. A 1.5

meter margin was used to establish "correctness" of a location estimate, corresponding with a small neighborhood of reference measurements.

Unless otherwise specified, the autocorrelation vectors were preprocessed as described in Section 3.1. Each test point was applied to the localization system, which returns a likelihood associated with every candidate point. Afterwards, a final localization estimate is chosen as the position with the highest likelihood of all candidate points, or

$$\hat{\boldsymbol{z}}_t = \boldsymbol{x}_{m^*}, \tag{4.9}$$

where

$$m^* = \arg\min_m \|\boldsymbol{r}_{\boldsymbol{z}_t} - \boldsymbol{r}_m\| \tag{4.10}$$

for the L_2 one-nearest neighbor method, and

$$m^* = \arg\max_m f_{q(\boldsymbol{x}_m)|\mathcal{Q}}(q_{\boldsymbol{z}_t}) \tag{4.11}$$

for the Gaussian random process method. Thus a single location estimate is produced for each test point.

A final performance score is then calculated by the proportion of test points localized correctly within the chosen margin of 1.5 meters. That is,

$$P = \frac{1}{T}\sum_{t=0}^{T-1} I_{d<1.5m}(\|\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t\|), \tag{4.12}$$

where

$$I_{d<1.5m}(d) = \begin{cases} 1, & d \leq 1.5 \text{ m} \\ 0, & d > 1.5 \text{ m} \end{cases} \tag{4.13}$$

This score increases as more location estimates are close to the corresponding test points. This value ranges from zero, when no points are localized correctly, to a value of one where all location estimates are within 1.5 meters of the corresponding test points. In all of tests discussed in this section, the data was produced with a sample rate which is double the

specified bandwidth, giving a 50% bandwidth utilization. Except for the results in 4.10, the Gaussian process method was used to evaluate fingerprinting performance.

Figures 4.10 — 4.16 illustrate fingerprinting performance in the environment described, under various scenarios and conditions. In these figures, the dashed black line indicates a baseline performance which would be obtained if grid points were selected at random from a uniform distribution (as even a random guess would lead to some proportion of accurate localizations). The x's in these plots represent performance for individual receiver locations, with the median performance of all receiver locations denoted by a line.

4.2.3 Gaussian process method compared to L_2 one-nearest-neighbor approach

In Figure 4.10 The Gaussian process localization performance (in blue) is compared to that of a simple L_2 one-nearest-neighbor approach, where the transmitter location estimate is simply the position corresponding to the reference autocorrelation which is closest in L_2 distance to the test autocorrelation.

At low bandwidths, only the L_2 one-nearest-neighbor method seems to have any benefit above baseline, and neither method improves significantly with bandwidth until a certain threshold is reached. This suggests that autocorrelation features only become informative once a minimum bandwidth is reached. As bandwidth increases and shape features become more intricate, the benefits of the Gaussian random process method begin to manifest. Until that point, a naive nearest neighbor approach is actually preferable, though it could be argued that the gains in this area are still too low to warrant usage. Ultimately the proportion of localizations that is accurate is relatively small, even at higher bandwidths. This suggests that the position can not be uniquely localized from one receiver in most situations. Since this is unacceptable to the requirements of most localization systems, it is recommended that multiple receivers be used, or that multiple measurements over time are leveraged (in the case of a moving transmitter).

4.2.4 Autocorrelation compared to channel impulse response

When the autocorrelation is used in place the full channel impulse response, the phase

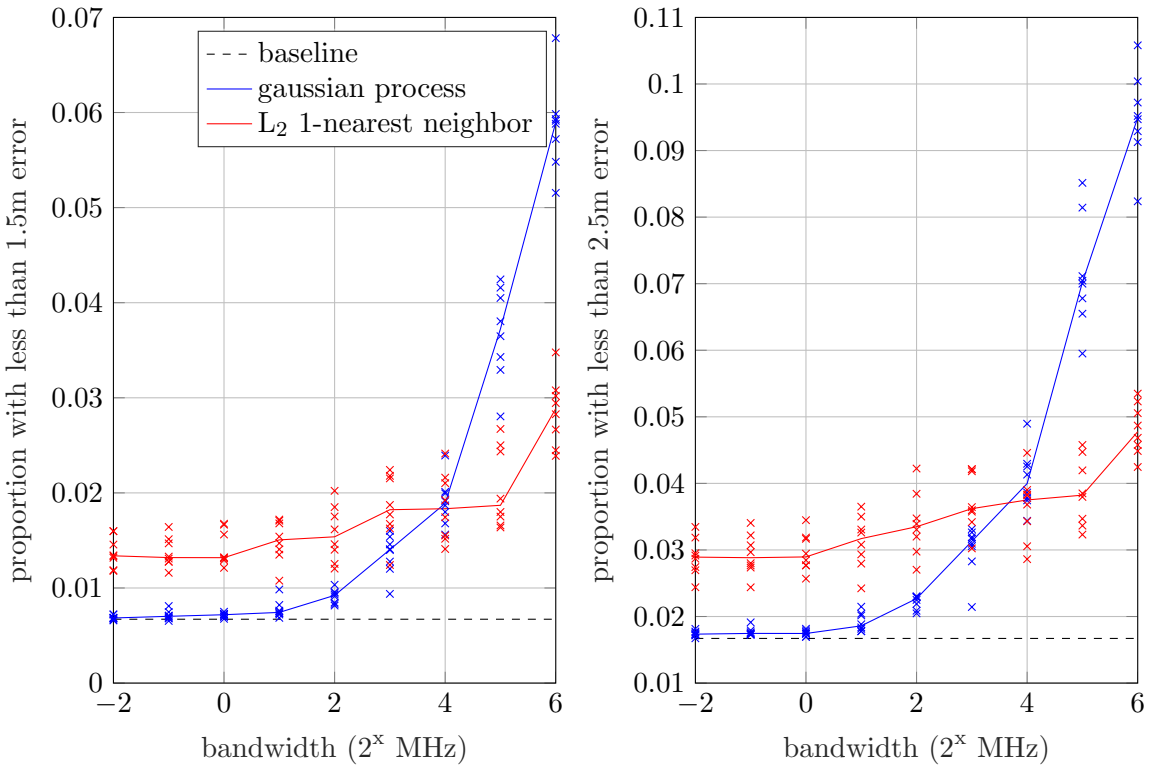


Figure 4.10: Performance comparison of Gaussian process and nearest-L_2 method

information about the original channel in the frequency domain is lost. This loss of information leads to a loss in performance, which can be observed in Figure 4.11, where the performance of fingerprinting using autocorrelation is compared to the performance using the channel impulse response. Here, the channel impulse response element magnitudes are used, to keep them real-valued. The reduction in performance from channel impulse response to autocorrelation is expected, and it is encouraging that a significant amount of performance is retained while using autocorrelation. If a system can estimate the channel impulse response, that definitely seems preferrable, but autocorrelation may be a viable alternative if channel estimation is not feasible.

4.2.5 Performance with different quantities of reference measurements

An important consideration when performing any fingerprinting method is the question of how much data is needed. In a grid situation, this often translates to the decision on the

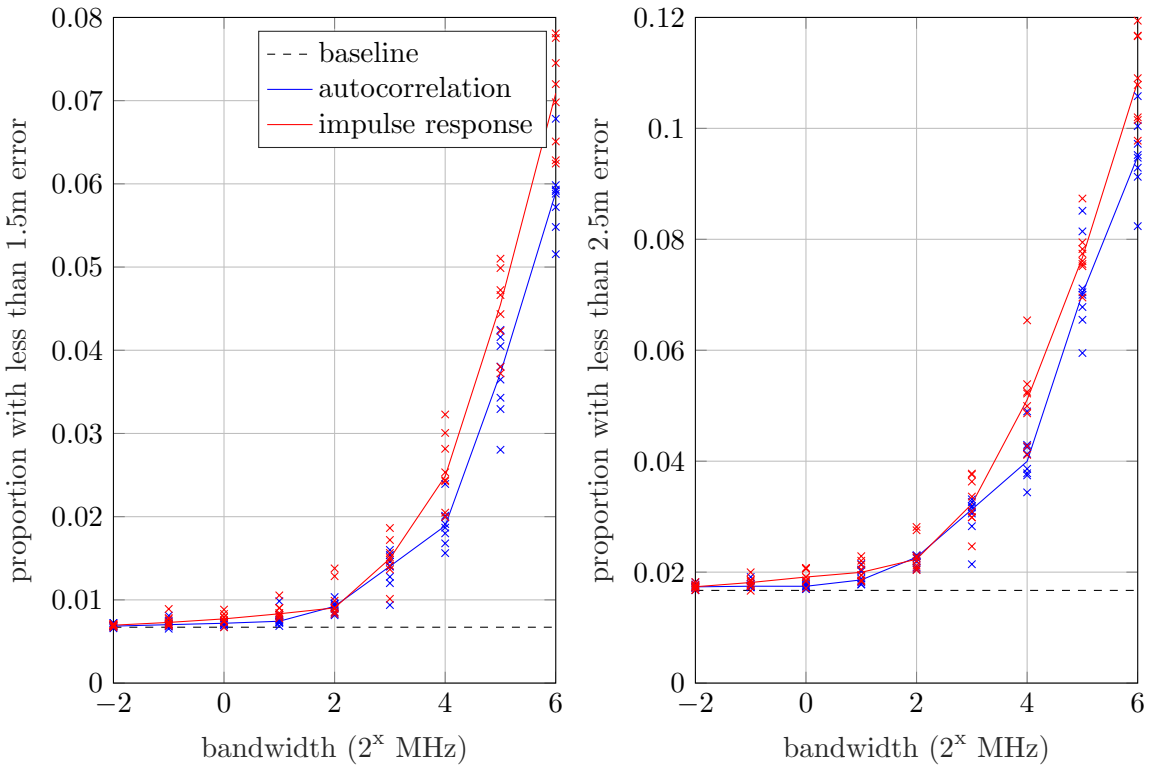


Figure 4.11: Performance comparison using autocorrelation versus the channel impulse response

spacing between reference measurements in this grid.

Figure 4.12 depicts the localization performance of the test system with the grid of reference measurements containing more points at different spacings (0.5 m, 2 m) along with the original 1 meter spacing results. In these other cases the candidate grid points were kept at the original 1 meter spacing of the original grid. Unsurprisingly, the performance of the system improves with more spatially-dense reference measurement sampling.

There is also something of an analog to the nyquist criterion, in that any changes which happen too quickly over distance will not only be lost with insufficient sampling, but will actually alias and obscure detectable, slower-changing phenomenon. As discussed in Section 4.1, there is likely to be some variation on the order of the signal wavelength or faster, particularly when moving axially around the receiver. This is not necessarily a strict a rule, as performance does not change radically as the wavelength changes even with

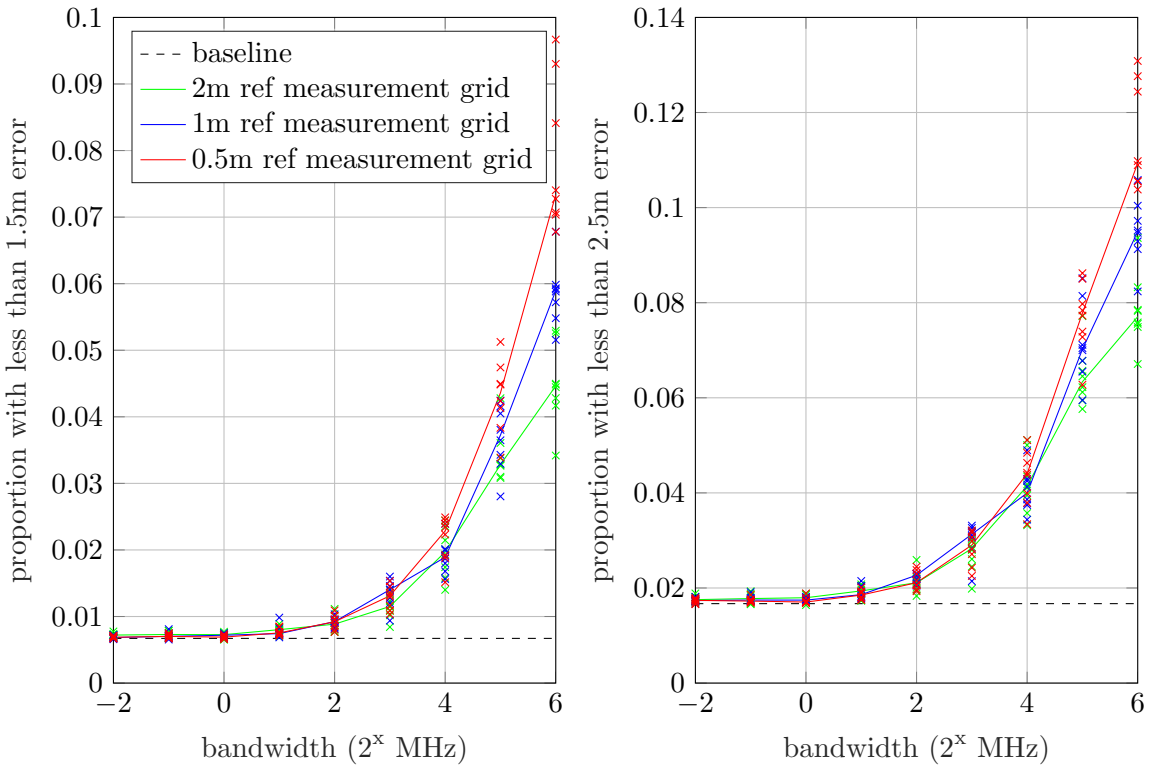


Figure 4.12: Localization performance with different reference measurement densities

a constant grid size. Unfortunately a more complete exploration into the question of how reference measuring sampling effects performance is beyond the scope of this research.

4.2.6 Performance with noise and limited measurements

The figures in other sections here are evaluated using autocorrelations which would be obtained if an infinitely long sample was used to calculate the autocorrelation. In practice, only limited amounts of data are available to calculate an autocorrelation for fingerprinting, and this autocorrelation is also influenced by receiver noise. This limitation is especially significant when the target signal is intermittent, or changing. In Figure 4.13, performance was evaluated as before, except that test autocorrelations are have errors as if they were calculated from a finite set of samples with noise. More specifically, the plot is simulating performance when 200,000 or two million samples are used, and with a median signal-to-noise ratio of 20dB and 40dB (SNR varies based on proximity to receiver, this SNR

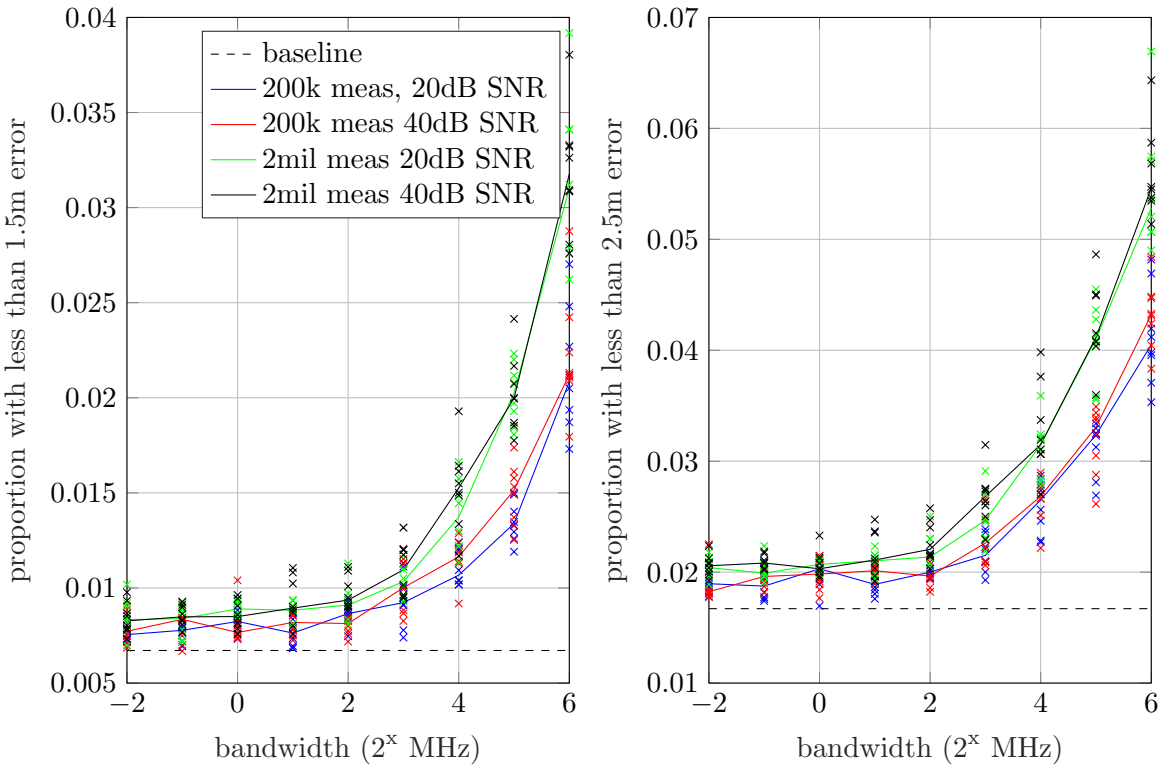


Figure 4.13: Localization performance with several combinations of data length and signal-to-noise ratio

is calculated as the median of this set of values). For details on how this is done, see Appendix A.2. As can be seen, performance improves as the SNR increases, and as the number of samples available decreases. These can be seen to significantly effect performance, so autocorrelation fingerprinting may not be appropriate if insufficient samples are available.

4.2.7 Performance with nonwhite localization signal

Other tests assume that the transmitted signals are spectrally white. While this can be controlled for the reference measurements, in an actual localization scenario, the signal is unlikely to be truly white. Many signals have a pulse shaping filter, which in turn influences the shape of the autocorrelation. In Figure 4.14 the results are shown for tests in which the test signal has been modified by a pulse shape like that which could result from a typical transmitted signal. These signals are chosen as a simple gaussian pulse shape and a square-root-raised cosine with 25% excess bandwidth. In these tests the reference signal is still

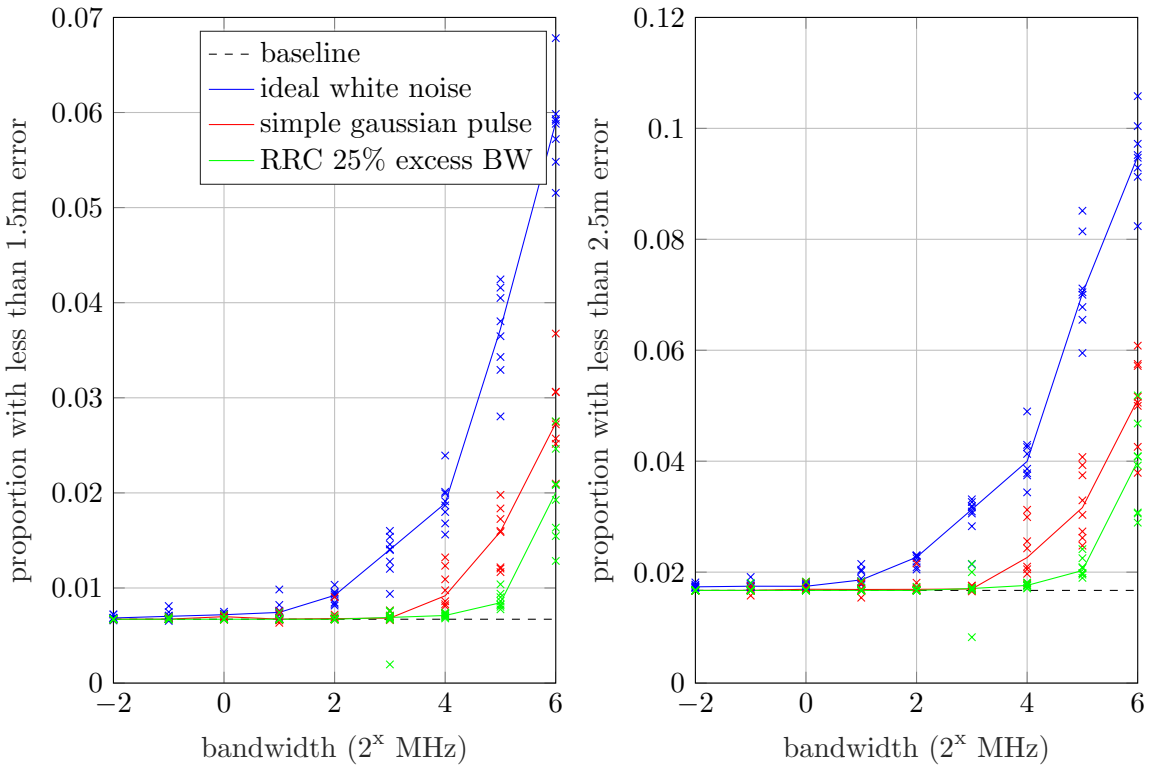


Figure 4.14: Localization performance with initial signal having a simple, but not ideal, autocorrelation pulse shape. Reference measurements remain based upon ideal source

assumed to have originated from a white noise reference, causing a discrepancy between the training measurements and test values. Even though performance degrades under these conditions, the method still provides some localization capability. Note that there is also a functional reduction in bandwidth caused by the change in pulse width, so that may be a contributor to the overall reduction in performance as well. It seems probable that if a signal was far enough from being spectrally white that it might not be localizable from autocorrelation fingerprinting. Signals which have time-varying autocorrelation functions could also cause difficulties for these methods.

4.2.8 Role of center frequency

The center frequency of the transmitted signal plays a major role in the relative phase of all received paths, as described in Section 4.1. This consequently effects the shape

of the channel impulse response and corresponding autocorrelation. The performance of fingerprinting at various center frequencies and bandwidths is shown in Figure 4.15. While

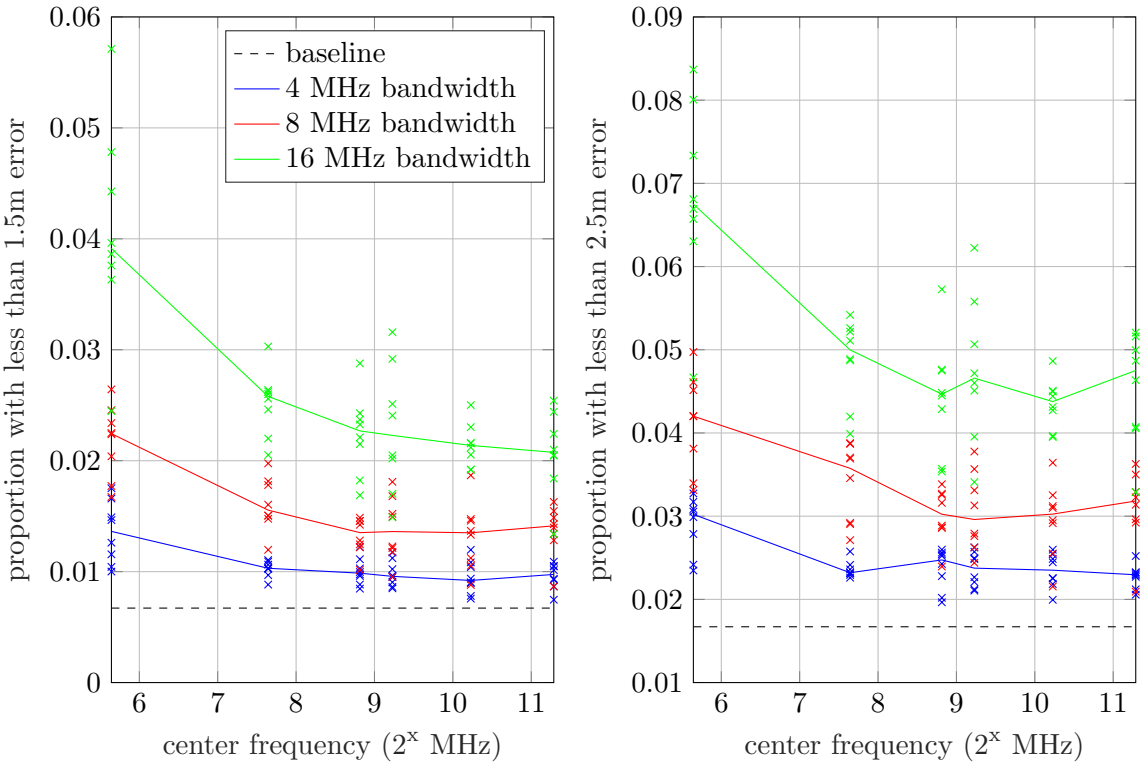


Figure 4.15: Performance of fingerprinting with regard to different center frequencies

performance is reduced at higher frequencies, this trend is much less significant than the corresponding trend for signal bandwidth. Note also that it is necessary for the reference measurements to be taken at approximately the same center frequency as the signal to be localized, or fingerprinting will not be effective.

4.2.9 Methods of handling complex phase

As discussed in Section 3.1.4, there are several ways of reducing a complex vector into a single real vector. In Figure 4.16 the performance of several of these options was compared. The methods depicted here include taking the magnitude of each element, concatenating a vector of the imaginary part to the real part, and taking the Fourier transform of the

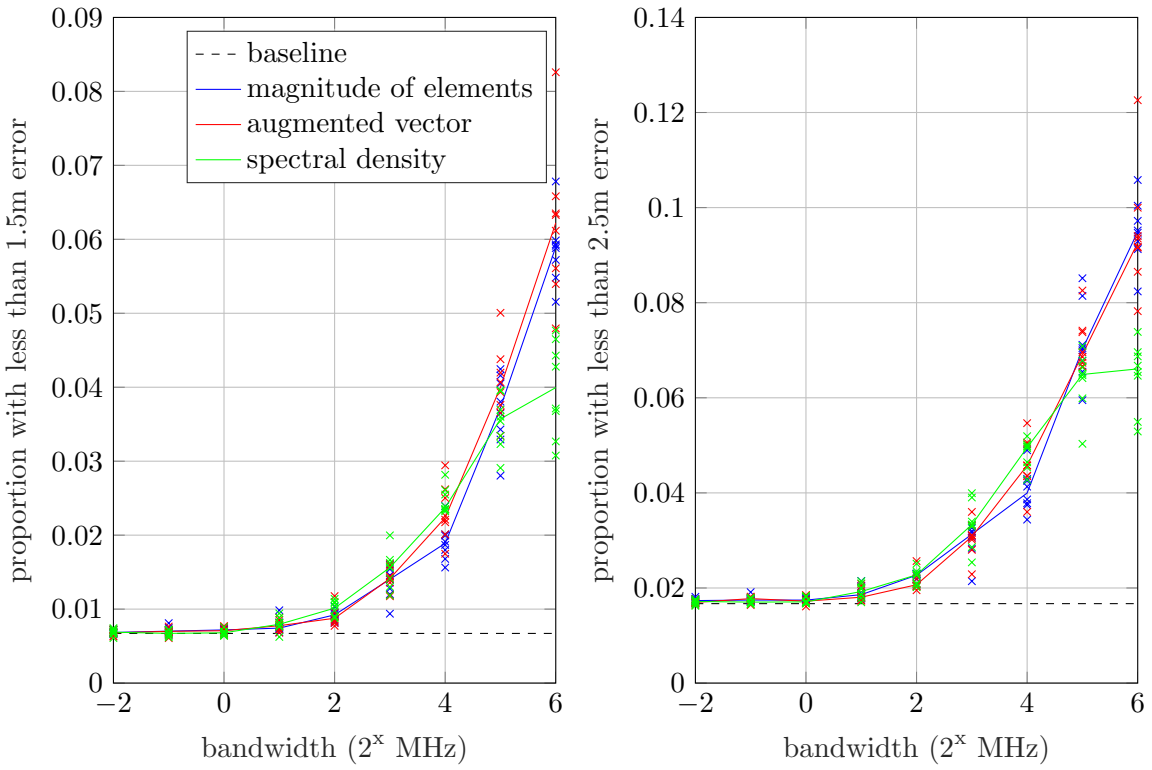


Figure 4.16: Performance of fingerprinting with different treatment of complex phase of elements

autocorrelation vector. The Fourier transform of the autocorrelation is the power spectral density, and is real and positively valued from the autocorrelation symmetry. Somewhat surprisingly, the augmentation and spectral density do not add any value of simply taking the magnitude of each element. This performance motivated the decision to use this treatment of complex phase throughout this research.

CHAPTER 5

Signal Propagation in an Indoor Environment

Source localization ultimately relies on the relationships between the transmitted signal, the received signal, and the environment. This section describes some fundamentals of these relationships. This includes some expected properties of a transmitted signal, the physical behavior of electromagnetic signals as they propagate through the environment, and additional effects of the receiver system. The overall process is summarized in Figure 5.1.

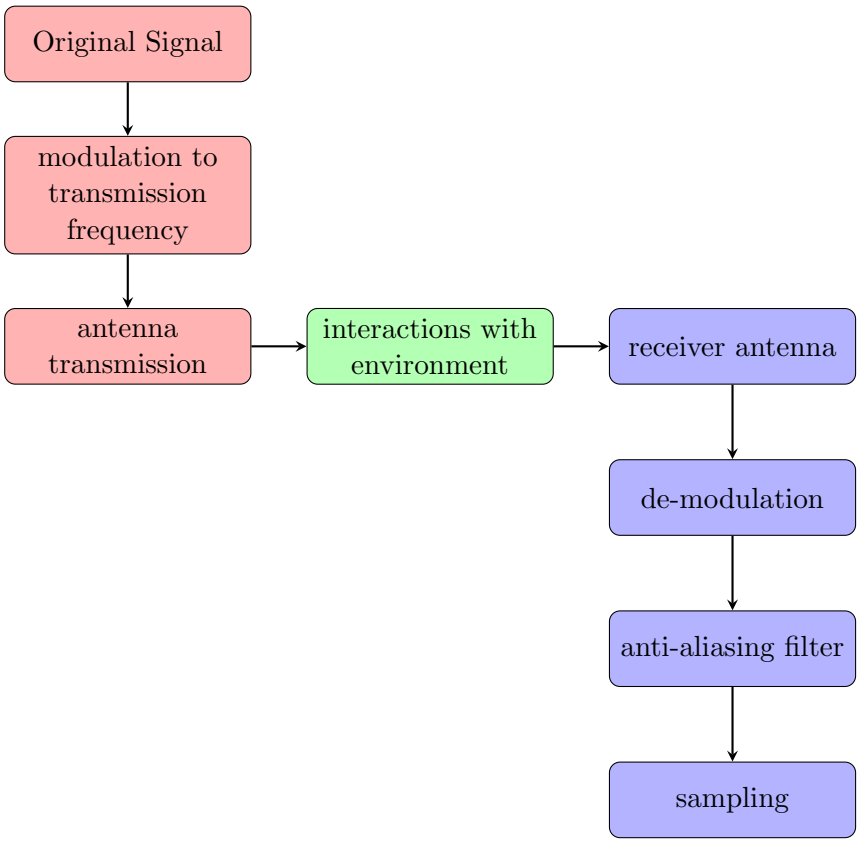


Figure 5.1: Description of transmitter-to-receiver relationship

5.1 Transmitted signal

Source localization may be desired for all kinds of RF signals. The effectiveness of various methods depends upon on differing assumptions about the signal of interest, and the available resources for localization. For example, some methods require prior information about the signal power. Some methods may even expect the transmitted signal to be known at the receiver. Autocorrelation-based fingerprinting, the topic of this thesis is a "blind" localization method, meaning that the original signal $s(t)$ need not be known by the localization system. That does not mean, however, that these methods are universal. As with all localization methods, autocorrelation-based fingerprinting relies on particular signal properties to function. The signal properties necessary to perform autocorrelation-based fingerprinting are discussed in this section along with relevant common properties of RF signals. Note that while many properties discussed are fairly common, some assumptions (particularly the spectral shape and bandwidth assumptions) will only apply to more specific situations. The effects of various stages of the complete transmitter-receiver system are summarized in Table 5.1 in a complex-valued form. While the complex-valued form is generally more practical, an alternative real-valued, vector-based form for most of these same equations is given in Table 5.2.

5.1.1 Separability

In order to effectively analyze a signal, it is necessary to be able to separate a signal from other signals. In many real-world RF systems, signals are designed with properties that allow for the separation of individual signals of interest. One common mechanism is utilizing different frequency regions for each signal. In this case, the basic processing of the receiver will adequately remove other signals. A particular signal of interest may also be the only one with significant power in a particular frequency region, such as when the transmitted signal is in an area where RF transmission is restricted. In other cases separating the signal may be more difficult, requiring more specialized methods. If interfering signals can not be completely removed, these signals will also contribute to noise at the receiver, negatively affecting performance. In this research, it is assumed that no interfering signals are present

in sufficient strength to cause noteworthy interference with the signal of interest, or that other signals have already been adequately removed.

5.1.2 Bandwidth and bandlimited signals

The bandwidth of a signal refers to the range of frequencies in which a signal has a nonnegligible amount of energy. Typical RF signals are functionally bandlimited in the electromagnetic frequency spectrum, meaning that their signal power is constrained to a particular set of frequencies $f \in [f_{\min}, f_{\max}]$.

This is bandlimitedness either by design, or just a natural result of the generation process. Since signal frequency affects physical propagation characteristics, many RF systems are designed to produce signals in particular frequency ranges. This is often achieved by mixing signals with a carrier frequency in a process called heterodyning (see Section 5.1.4). Even incidental RF signals are likely to have bandlimited characteristics, as any generated signals will follow frequencies naturally found in a system. Many electronic and physical interactions also have bandlimiting effects, simply by their tendency to pass some frequencies better than others. If signals are modelled as a random process, then bandlimitedness represents a statistical property of the signal. A final important factor is that even if a signal is not effectively bandlimited, the receiver processes perform filtering that will reject some frequencies of the signal, leading to a functionally bandlimited signal.

The effective bandwidth of a signal has a substantial effect on how much information it reveals about its environment, and correspondingly, the location of its source. These effects will ultimately affect system performance, and are observable in the results in Chapter 4. The bandwidth is also related to the timescale of the system, as discussed in Section 4.1.3. It is assumed here that the source signal is bandlimited and that the frequency region it covers is known, or can easily be estimated.

5.1.3 Autocorrelation shape

A signal which is bandlimited will likewise have a bandlimited autocorrelation function, which limits the shapes it can take. In this work, the shape of the autocorrelation function

estimated at the receiver is used to extract environment effects. It is not in general possible to separate structure found in the original signal autocorrelation function from the structure introduced by environmental effects. It is therefore assumed that the signal autocorrelation comes from a signal that is approximately spectrally white. This is likely to be at least approximately true if the signal is from a spectrally efficient communication system, and some signals are otherwise naturally spectrally white. These methods should also work if the environmental time scale is much larger than the time scale of the signal-of-interest, as illustrated in Figure 5.2. It may also be possible to estimate proper autocorrelation pulse shape if enough information is available at the receiver, but this is not explored.

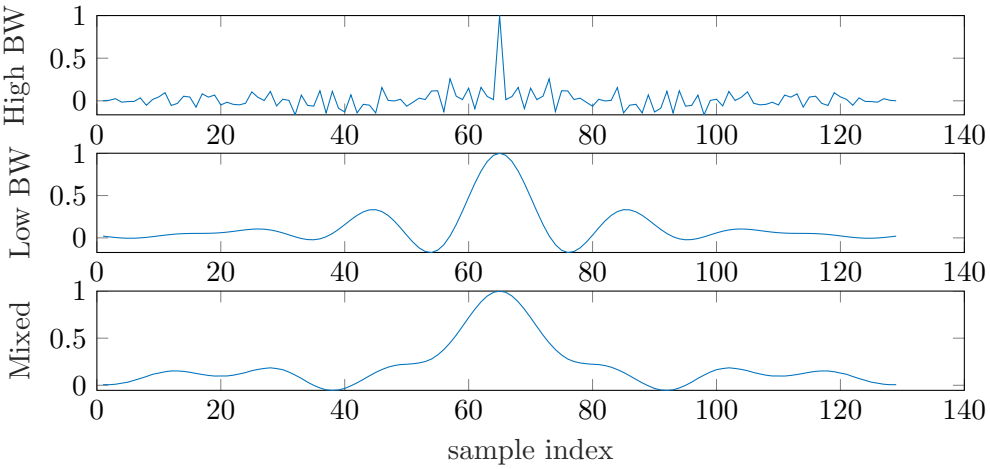


Figure 5.2: The autocorrelation of a short timescale (high bandwidth) input signal (top), compared to the autocorrelation of a long timescale (low bandwidth) channel impulse response (middle), compared to the autocorrelation of a the previous two convolved (bottom). Note that while the high bandwidth signal had a significant effect, the patterns in the long timescale channel contribution are still fairly visible. While this is not always the outcome, it is worth considering that environmental information may still be viewable in some cases.

5.1.4 Frequency and mixing

Signals are often generated by mixing a lower frequency, bandlimited signal with a sinusoidal signal at a higher frequency. This construction is typical of many digital communication systems (see Rice [37]). The signals are typically constructed by multiplying a pair

of signals (known as the in-phase and quadrature signals) with a carrier sinusoid. Letting $s_i(t)$ and $s_q(t)$ be the in-phase and quadrature signals, the the mixed signal is given by

$$s_{\text{mix}}(t) = s_i(t)\cos(2\pi f_c t) - s_q(t)\sin(2\pi f_c t) \tag{5.1}$$

.

This mixed signal is then be transmitted to the environment, where it will ultimately be captured in some form by the receiver. The base signal pair can also interpreted as a single complex signal, $s_{\text{baseband}}(t) = s_i(t) - js_q(t)$ which is what is typically referred to as the complex baseband signal. If the complex interpretation is used, then the mixed signal in (5.1) is equivalent to the real part of the signal multiplied by a complex exponential as

$$s_{\text{mix}}(t) = \Re[s_{\text{baseband}}(t)e^{j2\pi f_c t}] = \Re[(s_i + js_q)e^{j2\pi f_c t}]. \tag{5.2}$$

Using the definition of the baseband signal ($s_{\text{baseband}}(t) = s_i(t) - js_q(t)$) and expanding out the $\Re$ operation, this can also be expressed as

$$s_{\text{mix}}(t) = \frac{1}{2}s_{\text{baseband}}(t)e^{j2\pi f_c t} + \frac{1}{2}s_{\text{baseband}}{}^*(t)e^{-j2\pi f_c t}. \tag{5.3}$$

Even if a signal is not constructed this way, it can still be expressed as a modulated baseband signal. A bandlimited signal on any particular frequency range $f \in [f_{\min}, f_{\max}]$ can be expressed as a bandwidth range centered around a center frequency. Or written symbolically, $f \in [f_c - \frac{\beta}{2}, f_c - \frac{\beta}{2}]$, where $f_c = \frac{f_{\min}+f_{\max}}{2}$, and $\beta = f_{\max} - f_{\min}$. This is spectrally equivalent to some signal with $f \in [-\frac{\beta}{2}, -\frac{\beta}{2}]$, which is multiplied by the carrier $e^{-j2\pi f_c t}$. Since it is possible to interpret any bandlimited signal as a modulated baseband signal, every further source signal as some baseband signal modulated around a set carrier frequency f_c. As noted in Section 5.1.2, frequency affects propagation characteristics. In further sections, many assumptions and properties rely on "high frequency" signals. Usage of this term can vary between subject areas, but in this work "high frequency" refers to frequencies of at least 100 MHz, and typically greater than 300 MHz.

5.2 Receiver system

After a transmitted signal is produced, it propagates through the environment and passes through several processes at the receiver system. These effects must also be accounted for to understand the final received signal. The receiver system which includes an antenna, which can effect gain and may have a directional variation. Receiver systems are also influenced by receiver noise, distorting the received signal. After reception on the antenna, additional processing affects the signal. High frequency signals are typically mixed down to lower frequencies for further processing. Finally, digital receivers will have anti-aliasing filtering, quantization, and sampling effects as the continuous received signal is converted to a digital one.

5.2.1 Transmission through physical channel

As a signal propagates through an environment from a transmitter to receiver it can undergo a variety off effects. These interactions are the subject of Section 5.3, but in summary, signals are transferred through multiple paths, which are each affected by changes in amplitude, phase, and propagation time. At the receiver a linear combination of transmitted signals is received, at these differing amplitudes, phases and time delays. Additionally, there will be noise present in the receiver, as described in Section 5.2.2. If relevant environmental phase effects are incorporated into the complex amplitude, this can written as

$$y(t) = \sum_{m=0}^{M-1} \alpha_m s_{\text{mix}}(t - \tau_m) + w(t), \tag{5.4}$$

where $y(t)$ is the received signal, α_m is a complex amplitude, $s_{\text{mix}}(t)$ is the originally transmitted signal, τ_m is a propagation delay, $w(t)$ is a Gaussian white noise process, and M is the total number of received paths.

5.2.2 Receiver noise

At the receiver, the measured signal will include components produced by effects other than the received signal of interest. Receiver noise most commonly occurs due to heat in

the receiver. If other interfering signals are present at the receiver these signals will also contribute to the total noise. Fortunately, this noise is effectively white, meaning that its effects between times are uncorrelated. This property reduces some of the adverse effects of the noise.

5.2.3 Heterodyne or mixing

When signals of a high frequency are processed, they are typically mixed with a sinusoid to allow themm to be processed at lower frequencies. This is necessary to make sampling feasible for high frequency signals.

The received signal is mixed down by multiplying with a sinusoid, which ideally will allow for the recovery of a complex baseband signal, similar to what is often transmitted by RF systems. Ideally, the mixing frequency would match the original center frequency of the transmitted signal f_c. Since the receiver does not know f_c precisely (and it may even be slightly different due to doppler effects), it will mix with an approximate center frequency f_r which differs from f_c by some quantity Δf, related as

$$f_r = f_c - \Delta f$$

Mixing is then carried out by multiplying the corresponding complex sinusoid $\exp\left[-j2\pi f_r + \phi_0\right]$, where ϕ_0 is some unknown phase offset, and other variables are as described previously.

Doing this for one term of the sum in (5.4) (applying the time delay τ_m), and expanding $s_{\text{mix}}(t)$ as in (5.3) yields

$$s_{\text{de-mixed}}(t) = \frac{1}{2} s_{\text{baseband}}(t-\tau_m) e^{j2\pi(f_c-f_r)t - j2\pi f_c\tau_m + j\phi_0)} + \frac{1}{2} s^*_{\text{baseband}}(t-\tau_m) e^{-j2\pi(f_c+f_r)t + j2\pi f_c\tau_m + j\phi_0)}. \tag{5.5}$$

This is then rearranged in two ways. First, a substitution is performed using $\Delta f = f_c - f_r$. Second, an zero-sum term $j2\pi f_r\tau_m - j2\pi f_r\tau_m = 0$ is added to exponent on the first term. This lets the $-j2\pi f_c\tau_m$ term be combined as $-j2\pi f_c\tau_m + j2\pi f_r\tau_m = -j2\pi\Delta f\tau_m$, which

can then be combined with the $j2\pi\Delta f$ term, giving

$$s_{\text{de-mixed}}(t) = \frac{1}{2}s_{\text{baseband}}(t-\tau_m)e^{j2\pi\Delta f(t-\tau_m)-j(2\pi f_r\tau_m+\phi_0)}+\frac{1}{2}s^*_{\text{baseband}}(t-\tau_m)e^{-j2\pi(f_c+f_r)t+2j(\pi f_c\tau_m+j\phi_0)}. \tag{5.6}$$

This leads to a complete equation of

$$y(t) = \sum_m[\alpha_m\frac{1}{2}s_{\text{bb}}(t-\tau_m)e^{j2\pi\Delta f(t-\tau_m)}e^{-j(2\pi f_r\tau_m+\phi_0)}] + e^{-j(2\pi f_r t-\phi_0)}w(t) + \tilde{s}_{\text{hf}}(t), \tag{5.7}$$

where some higher frequency terms are grouped together as $\tilde{s}_{\text{hf}}(t)$, noting that this component (which is modulated at $f_c + f_r$) will later be removed through filtering.

5.2.4 Anti-aliasing filtering

After the received signal has been mixed down, it is typically passed through a low-pass filter to eliminate the high frequency terms from mixing, and avoid aliasing effects in the later sampling stage for discrete systems. For simplicity, it is assumed that high frequency components are eliminated entirely. These aliasing effects occur if the signal is not sampled fast enough to meet the Nyquist criterion ($f_s \geq \frac{f_{\max}}{2}$), which will cause signal distortion due to aliasing will occur. Typically the anti-aliasing filter will eliminate higher frequency components which would cause this criterion to be violated.

The anti-aliasing filter can be expressed using convolution as

$$y_{\text{filtered}}(t) = h_{\text{AAF}}(t) * \sum_m[\alpha_m\frac{1}{2}s_{\text{bb}}(t-\tau_m)e^{j2\pi\Delta f(t-\tau_m)}e^{-j(2\pi f_r\tau_m+\phi_0)}] + e^{-j(2\pi f_r t-\phi_0)}\tilde{w}(t), \tag{5.8}$$

where $h_{\text{AAF}}(t)$ is the impulse response of a low-pass anti-aliasing filter.

Note that in real systems, an additional intermittent mixing and filtering stage may be applied, however these will be a functionally equivalent to a single stage mixing and filtering system up to the effects of such systems that are within the scope of this discussion. Thus any effects of an intermittent stage are assumed here to have occurred as part of the later mixing and filtering steps.

By distributing the convolution over this sum, moving the time delay to the anti-aliasing filter, delay using properties of convolution, and letting $\tilde{\alpha}_m = \alpha_m e^{-j2\pi f_r \tau_m + \phi_0}$ this can be expressed as

$$y_{\text{filtered}}(t) = [\sum_m \tilde{\alpha}_m h_{\text{AAF}}(t - \tau_m)] * e^{j2\pi\Delta f t} s_{\text{baseband}}(t) + w(t) \tag{5.9}$$

5.2.5 Discrete-time digital systems

This work focuses on digital receiver systems. In these systems, the signals must ultimately be translated to sequences of discrete values. This digitization has several effects.

First the continous waveform $y_{\text{filtered}}(t)$ is sampled at discrete points, with a sampling period T between measurements. This is indicated by letting $t = nT$ in $y_{\text{filtered}}(t)$. When the noise term is sampled, it becomes a discrete-time noise process, which is also white.

$$y_{\text{filtered}}(n) = [\sum_m \tilde{\alpha}_m h_{\text{AAF}}(nT - \tau_m)] * e^{j2\pi\Delta f nT} \cdot s_{\text{baseband}}(nT) + w(n) \tag{5.10}$$

In addition to becoming a discrete-time signal, the values of the signal are quantized to digital representations, losing some precision. This quantization effect can be viewed as additional, uncorrelated system noise, and can often be simply interpreted as part of the receiver noise.

5.2.6 Final discrete linear model of channel

The final discrete system can be modeled as

$$y_{\text{filtered}}(n) = h_{\text{sys}}(n) * [e^{j2\pi\Delta f nT} \cdot s_{\text{baseband}}(nT)] + w(n), \tag{5.11}$$

where

$$\begin{aligned} h_{\text{sys}}(n) &= \sum_m \tilde{\alpha}_m h_{\text{AAF}}(nT - \tau_m) \\ &= e^{j\phi_0} \sum_m \alpha_m e^{-j2\pi f_r \tau_m} h_{\text{AAF}}(nT - \tau_m). \end{aligned} \tag{5.12}$$

Thus the effects of the transmission processed can be described by a sampling of the source signal $s_{\text{baseband}}(t)$ at $t = nT$, multiplication by sinusoid based on the transmitter-receiver frequency difference Δf, convolution with an environment transfer function $h_{\text{sys}}(n)$, and addition of a white noise signal $w(n)$. There is also a static phase offset $e^{j\phi_0}$, though this is not important to the work considered here.

5.2.7 Discrete autocorrelation

The autocorrelation of a signal is calculated as

$$r_y(k) = \frac{1}{N-k} \sum_{n=0}^{N-1} y(n) y^*(n-k). \tag{5.13}$$

If the scaling factor $\frac{1}{N-k}$ is removed, then this can be expressed as the convolution

$$\tilde{r_y}(k) = y(n) * y(-n). \tag{5.14}$$

Using (5.14) with the received signal from (5.11) and expanding terms gives

$$\begin{aligned} r_y(k) = [h_{\text{sys}}(n) * h_{\text{sys}}(-n)] * [s_{\text{bb}}(n) e^{j2\pi\Delta f nT} * s_{\text{bb}}(-n) e^{-j2\pi\Delta f nT}] \\ + [h_{\text{sys}}(n) * s_{\text{bb}}(n) e^{j2\pi\Delta f nT}] * w(-n) \\ + w(n) * [h_{\text{sys}}(-n) * s_{\text{bb}}(-n) e^{-j2\pi\Delta f nT}] \\ + w(n) * w(-n). \end{aligned} \tag{5.15}$$

This can be written as

$$r_y(k) = [e^{j2\pi\Delta f kT} r_{\text{sys}}(k)] * r_{\text{sig}}(k) + [r_{\text{cross}}(k) + {r_{\text{cross}}}^*(-k)] + r_{\text{w}}(k), \tag{5.16}$$

with

$$r_{\text{sys}}(k) = h_{\text{sys}}(n) * h_{\text{sys}}(-n) \tag{5.17}$$

$$r_{\text{sig}}(k) = s_{\text{bb}}(n) * s_{\text{bb}}(-n) \tag{5.18}$$

$$r_{\text{cross}}(k) = [h_{\text{sys}}(n) * s_{\text{bb}}(n)] * w(-n) \tag{5.19}$$

$$r_{\text{w}}(k) = w(n) * w(-n). \tag{5.20}$$

Note that the modulation factor $e^{j2\pi\Delta f nT}$ ends up modulating the autocorrelation as $e^{j2\pi\Delta f kT}$, and if Δf is small this will have only a small effect. If enough samples are used (length of convolution is sufficiently large), then $r_{\text{cross}}(k)$ will approach zero as the signal and noise terms should be uncorrelated, and $r_{\text{w}}(k)$ will approach $N_0^2\delta(k)$. Similarly, $r_{\text{sig}}(k)$ will approach the statistical autocorrelation of the transmitted baseband signal. If this signal is spectrally white, this term will approach r_{sys}, leaving only the channel information. These lead to an unscaled autocorrelation of the form

$$r_y(k) = r_{\text{sys}}(k) + N_0^2\delta(k). \tag{5.21}$$

The form found in (5.21) is used as the base equation for the autocorrelation fingerprinting in this thesis.

Table 5.1: Equations describing various stages of system effects (complex form)

Baseband signal	-	$s_{\text{baseband}}(t) = s_i(t) - js_q(t)$
Mixed signal	$R_{2\pi f_c t} \cdot x(t)$	$s_{\text{baseband}}(t)e^{j2\pi f_c t}$
Mixed signal - real part	$\Re[x(t)]$	$s_{\text{mix}}(t) = \Re[s_{\text{baseband}}(t)e^{-j2\pi f_c t}]$
Mixed signal - real part (b)	-	$s_{\text{mix}}(t) = \frac{1}{2}s_{\text{baseband}}(t)e^{j2\pi f_c t} + \frac{1}{2}s^*_{\text{baseband}}(t)e^{-j2\pi f_c t}$
Additive multipath at receiver	$\sum_m[\alpha_m\delta(t-\tau_m) * x(t)] + w(t)$	$y(t) = \sum_m \alpha_m s_{\text{mix}}(t-\tau_m) + w(t)$
De-mixed signal (individual term)	$R_{(-2\pi f_r t+\phi_0)} \cdot x(t)$	$s_{\text{de-mixed}}(t) = \frac{1}{2}s_{\text{baseband}}(t-\tau_m)e^{j(2\pi\Delta f t - 2\pi f_c\tau_m+\phi_0)} + \frac{1}{2}s^*_{\text{baseband}}(t-\tau_m)e^{-j(2\pi(f_c+f_r)t - 2\pi f_c\tau_m+\phi_0)}$
De-mixed signal (full)	-	$y_{\text{filtered}\,s}(t) = \sum_m[\alpha_m\frac{1}{2}s_{\text{bb}}(t-\tau_m)e^{j2\pi\Delta f(t-\tau_m)}e^{-j(2\pi f_r\tau_m+\phi_0)}] + e^{-j(2\pi f_r t-\phi_0)}w(t) + \tilde{s}_{hf}(t)$
Filtered signal	$h_{\text{AAF}}(t) * x(t)$	$y_{\text{filtered}}(t) = h_{\text{AAF}}(t) * \sum_m[\alpha_m\frac{1}{2}s_{\text{bb}}(t-\tau_m)e^{j2\pi\Delta f(t-\tau_m)}e^{-j(2\pi f_r\tau_m+\phi_0)}] + e^{-j(2\pi f_r t-\phi_0)}\tilde{w}(t)$
Filtered signal (altenative)	-	$y_{\text{filtered}}(t) = [\sum_m \tilde{\alpha}_m h_{\text{AAF}}(t-\tau_m)] * e^{j2\pi\Delta f t}s_{\text{baseband}}(t) + w(t)$
Sampled signal	$x(t)\|_{t=nT}$	$y_{\text{filtered}}(n) = [\sum_m \tilde{\alpha}_m h_{\text{AAF}}(nT-\tau_m)] * e^{j2\pi\Delta f nT}s_{\text{baseband}}(nT) + w(n)$
Autocorrelation	$x(n) * x(-n)$	$r_y(k) = r_{\text{sys}}(k) + N_0^2\delta(k)$

Table 5.2: Equations describing various stages of system effects (matrix/real form)

Baseband signal	-	$\begin{bmatrix} s_i(t) \\ s_q(t) \end{bmatrix}$
Mixed signal	$R_{2\pi f_c t} \cdot x(t)$	$R_{2\pi f_c t} \begin{bmatrix} s_i(t) \\ s_q(t) \end{bmatrix}$
Mixed signal - real part	$\Re[x(t)]$	$s_{\text{mix}}(t) = s_i(t)\cos(2\pi f_c t) - s_q(t)\sin(2\pi f_c t)$
Mixed signal - real part (b)	-	$\begin{bmatrix} s_{\text{mix}}(t) \\ 0 \end{bmatrix} = \frac{1}{2} R_{2\pi f_c t} \begin{bmatrix} s_i(t) \\ s_q(t) \end{bmatrix} + \frac{1}{2} R_{-2\pi f_c t} \begin{bmatrix} s_i(t) \\ s_q(t) \end{bmatrix}$
Additive multipath at receiver	$\sum_m [\alpha_m \delta(t-\tau_m) * x(t)] + w(t)$	$y(t) = \sum_m \alpha_m s_{\text{mix}}(t-\tau_m) + w(t)$
De-mixed signal (individual term)	$R_{(-2\pi f_r t + \phi_0)} \cdot x(t)$	$\begin{bmatrix} s_{\text{de-mixed}_i}(t) \\ s_{\text{de-mixed}_q}(t) \end{bmatrix} = \frac{1}{2} R_{(2\pi \Delta f t - 2\pi f_c \tau_m + \phi_0)} \begin{bmatrix} s_i(t-\tau_m) \\ s_q(t-\tau_m) \end{bmatrix} + \frac{1}{2} R_{(-2\pi(f_c+f_r)t - 2\pi f_c \tau_m + \phi_0)} \begin{bmatrix} s_i(t-\tau_m) \\ s_q(t-\tau_m) \end{bmatrix}$
De-mixed signal (full)	-	$\begin{bmatrix} y_i(t) \\ y_q(t) \end{bmatrix} = \sum_m (\alpha_m \frac{1}{2} R_{2\pi \Delta f (t-\tau_m)} R_{(-2\pi f_r \tau_m + \phi_0)} \begin{bmatrix} s_i(t-\tau_m) \\ s_q(t-\tau_m) \end{bmatrix}) + \begin{bmatrix} \cos(2\pi f_r t - \phi_0) \\ -\sin(2\pi f_r t - \phi_0) \end{bmatrix} w(t) + \tilde{s}_{hf}(t)$
Filtered signal	$h_{\text{AAF}}(t) * x(t)$	$\begin{bmatrix} y_i(t) \\ y_q(t) \end{bmatrix} = h_{\text{AAF}}(t) * \sum_m (\alpha_m \frac{1}{2} R_{2\pi \Delta f (t-\tau_m)} R_{(-2\pi f_r \tau_m + \phi_0)} \begin{bmatrix} s_i(t-\tau_m) \\ s_q(t-\tau_m) \end{bmatrix}) + \begin{bmatrix} \cos(2\pi f_r t - \phi_0) \\ -\sin(2\pi f_r t - \phi_0) \end{bmatrix} \tilde{w}(t)$
Filtered signal (altenative)	-	$\begin{bmatrix} y_i(t) \\ y_q(t) \end{bmatrix} = [\sum_m A_m h_{\text{AAF}}(t-\tau_m)] * R_{2\pi \Delta f t} \begin{bmatrix} s_i(t) \\ s_q(t) \end{bmatrix}) + w(t)$
Sampled signal	$x(t)\|_{t=nT}$	$\begin{bmatrix} y_i(n) \\ y_q(n) \end{bmatrix} = [\sum_m A_m h_{\text{AAF}}(nT-\tau_m)] * R_{2\pi \Delta f nT} \begin{bmatrix} s_i(nT) \\ s_q(nT) \end{bmatrix}) + w(n)$

*Using $A_m = \frac{1}{2}\alpha_m R_{(-2\pi f_r \tau_m + \phi_0)}$

5.3 Electromagnetic wave propagation and ray model

The properties of electromagnetic signal propagation can be described fundamentally by Maxwell's equations, which form the basis of classical electromagnetics. The solutions to these equations describe the full interaction of a transmitting antenna on an environment and ultimately a receiving antenna. In general cases, solving these equations directly is analytically intractable, and therefore requires numerical solutions or alternative methods.

When dealing with radio transmissions at sufficiently high frequencies, one can model the wave propagation as rays emitted radially from the transmitter. The rays interact with surfaces and, when incident on a receiver antenna, lead to a measured received signal. This model leads to the ray-tracing or ray-casting method of simulating an electromagnetic environment. This method is generally more computationally tractable than solving Maxwell's equations directly, particularly for complex environments, and can still produce accurate descriptions of environmental effects on the signal. Ray-tracing is a method used to model radio-wave propagation, particularly in urban and indoor environments.

5.3.1 Transmitter-to-receiver interaction

When ray-tracing is used to model of the environment, the behavior of these rays follows from the behavior and properties of electromagnetic waves. Typical waves will reflect, refract, and diffract in response to walls and other surfaces, and rays likewise inherit these behaviors. To model the relationship between a transmitter and receiver, the paths of rays are drawn from the transmitter, and followed through a series of these wall interactions until they reach the receiver. The signal from the transmitter will follow each of these paths, which will have a different path length, and also have a different associated attenuation and arrival phase. The combination of the signal due to each of the paths appears at the receiver as

$$y(t) = \sum_{m=0}^{M-1} \alpha_m s(t - \tau_m), \tag{5.22}$$

where $y(t)$ is the pre-noise received signal, $s(t)$ is the transmitted signal, and α_m and τ_m

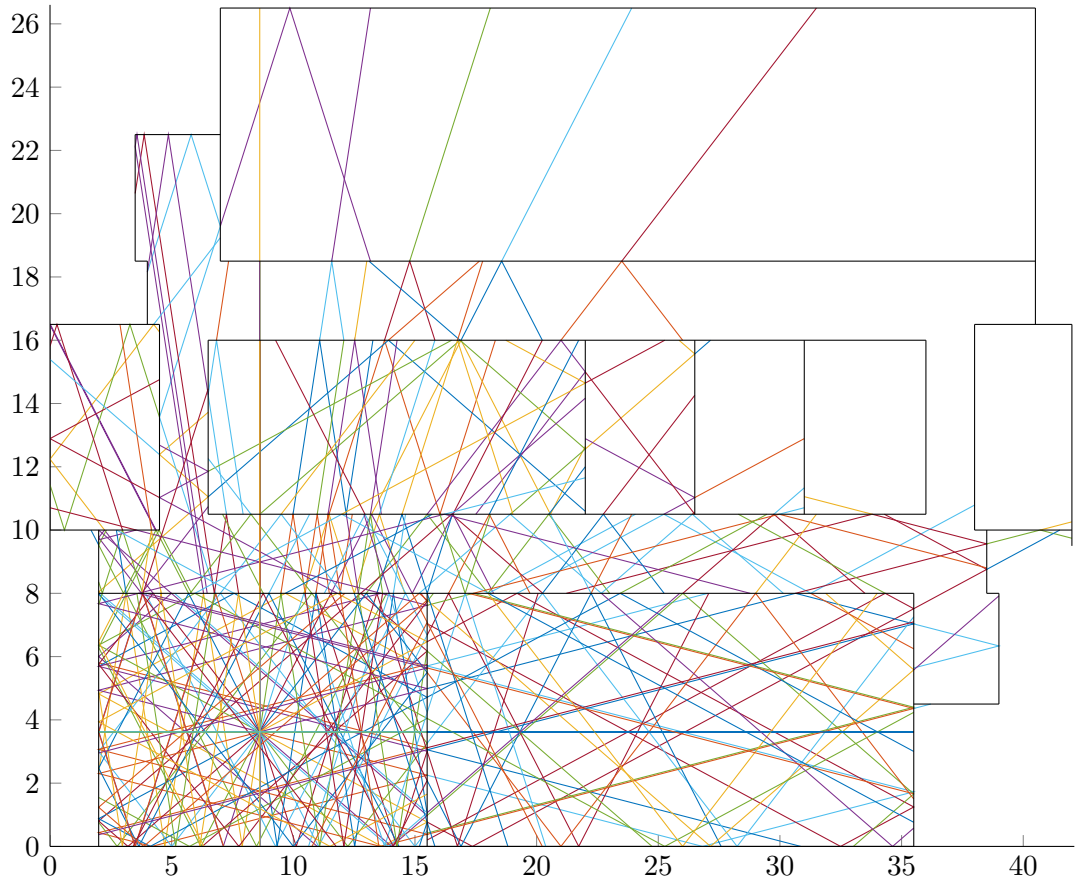


Figure 5.3: Ray-tracing model interprets elecromagnetic wave propagation using rays

are the path amplitudes and delays for a given path m. The previous equation 5.4 follows fairly directly from this (except with added noise).

It is worth noting that $y(t)$, $s(t)$, and α_m can be interpreted as either real-valued or complex-valued. If these quantities are viewed directly in terms of the electromagnetic field strength or current over time, they are real quantities. However most RF systems use signals which are modulated to higher frequencies (see Section 5.1.4). The structure of such a signal allows for a complex interpretation, which is often more practical. When complex signals are used, the complex phase of coefficients α is given by phase, discussed in Section 5.3.3

5.3.2 Antenna gains

The antenna plays a role at both the transmitter, and receiver. The transmission antenna dictates the polarization of the transmitted signals, which influences how it interacts with the environment, and ultimately the receiver antenna. Antennas have an associated gain value, given as G_T for a transmitting antenna and G_R for a receiving antenna. Additionally, antennas can have differing gains at different frequencies. This can apply a different gain/attenuation profile at different frequency components of the received signal, distorting the signal. Finally, many antennas have direction-dependent gains for either transmission or reception. This will influence the gains for different paths, as paths will typically be transmitted or received at different angles. For the purposes of this research, it is assumed that these effects are negligible, a value of one is used for G_R and G_T. In practice, the importance of these effects will depend on the antennas and frequencies involved in a particular setting.

5.3.3 Signal phase

As noted previously, modulated signals have structure which lead to a complex interpretation. This is perhaps most easily seen in the case of a sinusoid, which have a clear phase at any point in time and space. However phase also applies to signals which are modulated around a center frequency, as discussed in Section 5.1.4. When such a signal arrives at a transmitter, the resultant phase is a function of both the original transmission time, and the time required to propagate through the environment. Under multipath conditions, the signals are transmitted as one coherent signal, but the differing path lengths lead to different travelling times and therefore the signals arrive with different phases. The phase effects of the paths are related to path distance by:

$$\phi(t) = \exp j2\pi f_c(t - \frac{d}{c}), \tag{5.23}$$

which follows simply from the time delay of transmission. Note how this depends on the center frequency. For sinusoids this center frequency is simply the frequency of the sinusoid

itself. In other signals, a modulated structure may influence the role that signal phase has on a system, which can be seen throughout the equations in Section 5.1 and Section 5.2.

5.3.4 Signal polarization

Another characteristic of RF wave propagation is the wave polarization. This refers to the configuration of the electric and magnetic fields within the propagating wave. In most cases, this reduces simply to the physical orientation of the electric and magnetic field. These polarizations are typically described by the orientation of the electric field with regard to a relevant ground plane, as vertical or horizontal polarization, or with regard to its orientation to a surface it is interacting with, as "p" (parallel) or "s" (perpendicular) polarization.

Note that in RF scenarios, vertical and horizontal polarization often follow the general orientation of the transmitting antenna, hence vertical for antennas oriented vertically, and horizontal for antennas oriented horizontally. Note that it is possible for signals to contain multiple wave polarizations, as occurs in blackbody radiation due to material temperature. Polarization tends to be less mixed when emitted from an RF antenna, however.

Interactions between waves and walls are affected by wave polarization, as described in Section 5.3.6. Note that interactions with an ideal, vertical wall do not change the polarization of light, though they do attenuate different polarizations differently. Still, interactions with rough surfaces and compex objects at diffent angles will ultimately affect the polarization of a signal as it propagates through a real environment.

5.3.5 Path loss

As signals propagate through space, the signal spreads and decreases in power density. This is most simply modelled by the Friis Transmission Equation, which models this loss as

$$P_{\text{received}} = G_T G_R (\frac{\lambda}{2\pi d^{\gamma/2}})^2. \tag{5.24}$$

In this equation, P_{received} is the received power, λ is the signal wavelength, d is the distance

travelled by the signal between transmitter and receiver, γ is the loss coefficient, and G_T, G_R are the antenna gains for the transmitter and receiver, respectively. For free space, γ has a value of two. This can be adjusted to model propagation in different environments. This γ value should only be adjusted for environmental properties that aren't otherwise modelled, so when the environment is fully modelled, an γ of two is still appropriate. The antenna gains are dependent on the involved antennas, as discussed in Section 5.3.2.

5.3.6 Wall interaction

When a signal is inciedent upon an interface between two media, such as between the air and a building material, it can undergo reflection, refraction, and/or diffraction. This is depicted in Figure 5.4. In many cases, the object will be thin relative to the scale of the environment, and may contain multiple layers of different materials. With minimal loss of accuracy, one can assume that the non-air surfaces will be of negligible thickness, with a simple reflection/refraction profile. The reflection/refraction profile can be generated to characterize more complicated/multi-layer substances as well as noted in Section 5.3.7.

Dielectric properties

The wall materials used have a significant effect on how rays interact with them. These materials have various properties affecting different aspects of these interactions. The number of parameters needed to characterize a materials behavior depends on the types of materials used (for example metal walls, versus glass windows), and the accuracy to which they are modelled. While full analysis of electomagnetic properties of materials are beyond the scope of this research, some typical properties include the following:

- μ - Magnetic permeability, describes material response to magnetic fields
- ϵ - Electric permittivity, describes material response to electric fields
- σ - Conductivity, describes electron/current mobility
- ρ - Ruggosity, describes surface roughness

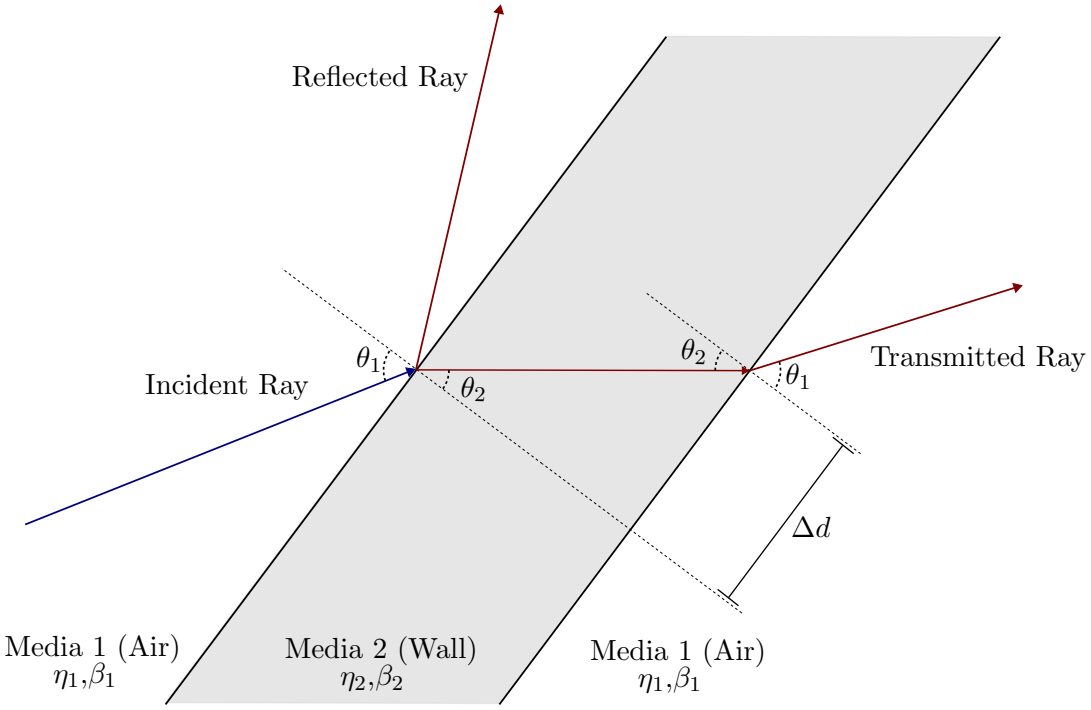


Figure 5.4: The interaction between an electromagnetic "ray" and a wall made of some other media

- η - Intrinsic impedance, given by $\eta = \sqrt{\frac{j\omega\mu}{\sigma+j\omega\epsilon}}$, or $\eta = \sqrt{\frac{\mu}{\epsilon}}$ for lossless materials (materials are approximately lossless when $\sigma \ll \omega\epsilon$, where ω is angular frequency of the signal.)

Some of these values are used in the basic equations regarding wall interactions, which will be used in later sections. Identifying proper values for these functions is the subject of investigation and research (such as by Regmi [38]). Some of these properties also have some dependence on frequency. It is assumed here that this variation will not be substantial within the frequency range of a particular signal, though it is necessary to select parameters which apply to a frequency near to the center frequency of the signal of interest. Some values of these parameters for common building materials is included in Table 5.3.

Table 5.3: Parameters for common wall materials

Parameter	Concrete floor/ceiling	MDF wall
μ^*	$4\pi \cdot 10^{-7}\ H \cdot m^{-1}$	$4\pi \cdot 10^{-7}\ H \cdot m^{-1}$
$\epsilon^\dagger$	$6.9\epsilon_0$	$60\epsilon_0$
$\sigma^\dagger$	$0.0138\ S \cdot m^{-1}$	$0.02\ S \cdot m^{-1}$
$\rho^\ddagger$	0.8	0.8
η_{lossless}	143.4 Ω	48.6 Ω

* Typical value for most common materials which are not ferromagnetic

† From Navarro et al. [39]

ϵ_0 is the permittivity of free space, equal to $8.854 \cdot 10^{-12} F \cdot m^{-2}$

‡ Generic value for walls given by Pahlavan and Levesque [36]

Reflection

An electromagnetic wave incident on a surface produces a reflection at the same angle of incidence, as described by Snell's law of reflection:

$$\theta_{\text{incident}} = \theta_{\text{reflected}}. \tag{5.25}$$

The reflected signal is also attenuated from the original signal by a factor R (known as the reflection coefficient) according to:

$$\alpha_{\text{reflected}} = \rho R_* \alpha_{\text{incident}}, \tag{5.26}$$

where α_{incident} is the amplitude of the incident signal, $\alpha_{\text{reflected}}$ is the amplitude of the received signal, ρ is the surface ruggosity, and $R_* = R_\perp$ or $R_\parallel$ depending on the wave polarization. These are given by

$$R_\perp = \frac{\eta_2 \cos\theta_i - \eta_1 \cos\theta_t}{\eta_2 \cos\theta_i + \eta_1 \cos\theta_t}, \tag{5.27}$$

$$R_\parallel = \frac{-\eta_1 \cos\theta_i + \eta_2 \cos\theta_t}{\eta_1 \cos\theta_i + \eta_2 \cos\theta_t} \tag{5.28}$$

where $R_\perp$ corresponds to perpendicular or s-polarized waves, and $R_\parallel$ corresponds to parallel

or p-polarized waves, θ_i and θ_t are the incident and transmitted angles (the transmitted angle is given by (5.29)), and η_1 and η_2 as the intrinsic impedances of the two media. They are known as reflection coefficients. In ray tracing, an additional reduction to reflected signals occurs because surfaces are not perfectly smooth. This loss is modelled simply as a constant multiplier ρ known as ruggosity. It can be convenient to absorb the ruggosity factor into the reflection coefficient as $R_{\text{full}} = \rho R_*$.

Depending on the materials and orientations of environmental surfaces, the interaction may have different phase effects. In cases where the antenna is vertically polarized in a building and interacting with vertical walls, where the the signal frequencies are high enough, it is generally suitable to assume that the phase change is simply a 180° phase change or sign flip. This is discussed further in Pahlavan and Levesque [36], and Balanis [40].

An example of final reflection and transmission coefficients for a slab of glass is depicted in Figure 5.5. Note that in the case of a slab, the equations must be used for both entering and exiting the slab media, with a differing angle of incidence inside the media. This profile also includes contributions from paths that reflect several times internally before leaving the media.

Transmission and Refraction

In addition to the reflected signal, a portion of the signal will continue through the media with the angle changed according to Snell's law of refraction:

$$\beta_1 \sin\theta_{\text{incident}} = \beta_2 \sin\theta_{\text{transmitted}}, \tag{5.29}$$

where β is the phase constant, given by $\beta = \omega\sqrt{\mu\epsilon}\sqrt{\frac{1}{2}[\sqrt{1+(\frac{\sigma}{\omega\epsilon})^2}+1]}$, or $\beta = \omega\sqrt{\mu\epsilon}$ for lossless media.

The transmitted signal amplitude is also modified according to a transmission coeffecient T as:

$$\alpha_{\text{transmitted}} = T_* \alpha_{\text{incident}}, \tag{5.30}$$

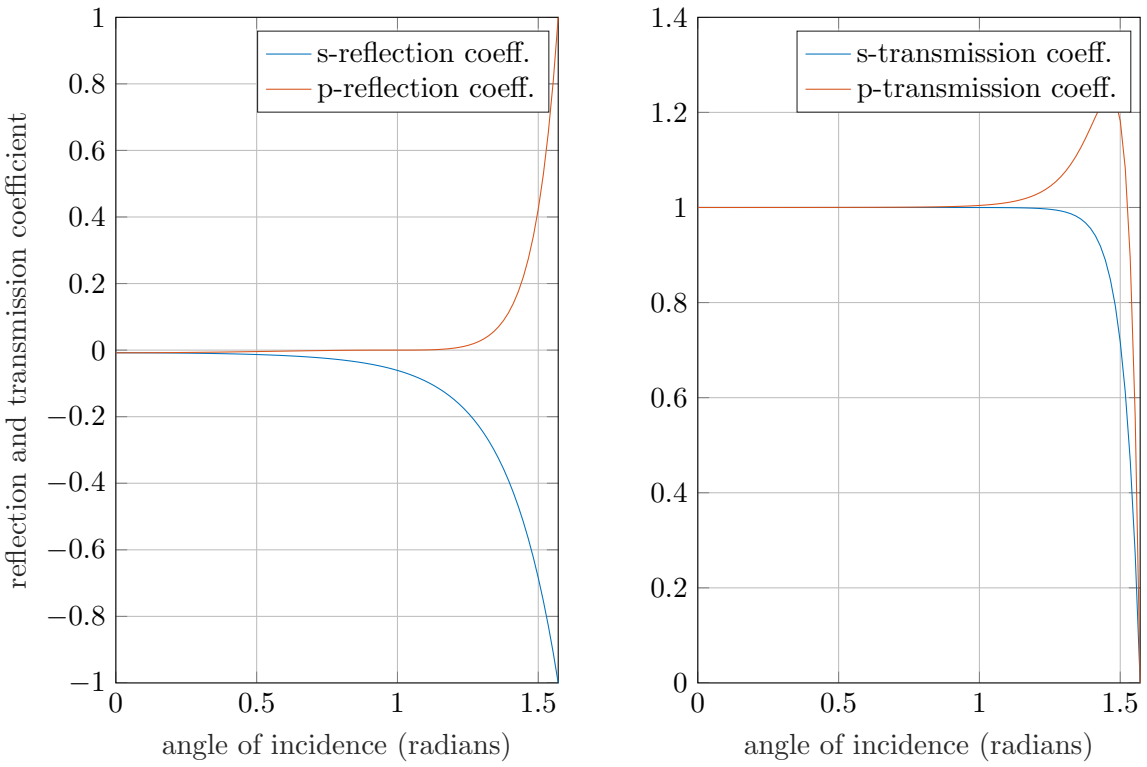


Figure 5.5: Reflection coefficients (Left) and transmission coefficients (Right) for a slab of glass$\epsilon_r = 1.5^2$.

where α_{incident} is the amplitude of the incident signal, $\alpha_{\text{transmitted}}$ is the amplitude of the signal transmitted past the medium, and $T_* = T_\perp$ or $T_\parallel$ depending on the wave polarization, as before. These are given by

$$T_\perp = \frac{2\eta_2 \cos\theta_i}{\eta_2 \cos\theta_i + \eta_1 \cos\theta_t} \tag{5.31}$$

$$T_\parallel = \frac{2\eta_2 \cos\theta_i}{\eta_1 \cos\theta_t + \eta_2 \cos\theta_t} \tag{5.32}$$

For interaction with a wall, the signal will return to the original media (air) after passing through all layers. This results in a distance offset as shown in Figure 5.4, but will return to the original propagation angle. If the wall of this distance is thin, it is small and may be neglected

When dealing with the wall as whole, the gain for the transmitted signal can also be found from the reflected power as

$$T_* = \sqrt{1 + R_*^2}. \tag{5.33}$$

This equation follows from conservation of energy, though this does assume no loss of power in the wall medium, which is a reasonable assumption in many, but not all cases. More details are discussed in Pahlavan and Levesque [36].

5.3.7 Multi-layer interfaces and overall modelling

When a medium has multiple layers, a combination of reflections and transmissions with the various layers occurs at every interface. These can all be modelled seperately, but it is common to combine the total reflection effect into a single reflection coefficient, and likewise for the transmission coefficient. Such models are often called slab models; these models are not used here, but equations for using them can be found in [36].

More complex methods of computing transmission and reflection coefficients can be also be performed without necessarily affecting the performance of ray-tracing simulations (like in Chapter 6). This is true as long as the coefficients can be distilled into a single

function of interaction angle.

Diffraction

Unlike reflection and refraction, diffraction occurs on corners and edges of media. As frequency increases, effects due to diffraction become increasingly negligible. The effects of signal diffraction were assumed to be minimal and were not accounted for in this research, though they are considered in some research (such as Navarro et al. [39]). If localization is being performed using lower frequency signals, then these effects should be considered more carefully. Alves et al. [41] and Pahlavan and Levesque [36] include equations describing diffraction, these can be referenced if needed for modeling propagation environment.

CHAPTER 6

Ray-tracing algorithm

6.1 Overview and variables

With the ideas developed from the previous section, an algorithm for modeling propagation in an environment using ray-tracing (also called "ray-casting") can be given.

This algorithm consists of two primary steps. The first is the simulation of rays propagating out from a receiver throughout a walled environment. The second step is the comparison of these ray paths to receiver locations to identify interactions. These two steps produce a list of path lengths and associated gains between a transmitter and any number of receivers. The algorithm is summarized in Algorithm 6.1 and Algorithm 6.2.

6.1.1 Comparison to other implementations

Note that there are a number of variations of this ray-tracing approach in handling rays, wall-interactions, etc. This section is meant to both provide an elementary example of ray-tracing, and document the ray-tracing methods used to generate the results provided in Chapter 4. Many other ray-tracing algorithms exist, such as by Hosseinzadeh [42]. The development of increasingly accurate and efficient ray-tracing algorithms is an area of ongoing area of research (such as by Geok et al. [43] and Navarro et al. [39]), though it is not a focus of this thesis.

6.1.2 Variables overview

In this implementation, geometric rays are given by $\{\boldsymbol{x} + \boldsymbol{v}t \mid t \geq 0\}$ with the variable parameters $\boldsymbol{x}$ as the source vertex, $\boldsymbol{v}$ as a vector giving the ray propagation direction, and t as parameter for a position on the resulting ray. Wall segments are likewise described as $\{s\boldsymbol{u}_0 + (1-s)\boldsymbol{u}_1 \mid 0 \leq s \leq 1\}$, or the set of points along a line segment between two vertices

$\boldsymbol{u}_0$ and $\boldsymbol{u}_1$.

Paths are stored as a sequence of position vertices $\{\boldsymbol{x}_n^{[p]}\}$, along with a vector giving the propagation direction $\{\boldsymbol{v}_n^{[p]}\}$, a wall collision sequence $\{\omega_n^{[p]}\}$, and a sequence of path gains $\{g_n^{[p]}\}$. For each ray n an initial propagation step p_0 is given which indicates what propagation step a particular ray was introduced.

The number of rays doubles at each propagation iteration, so the initial size N_0 must be set so that the final ray count $N = N_0 \cdot 2^P$ will not exceed memory constraints. Higher values of N and P increase model fidelity and therefore accuracy, though they give diminishing returns and increase computation time. The lookup table size L is fairly flexible, though a value of 90 is likely sufficient for most cases. The total number of walls W is mostly dependent on the environment complexity, with more walls coming at increased computational cost. A receiver collision radius d_{rx} must also be set, which determines how closely a ray must pass by a receiver to be registered. Higher values can lead to spurious receptions and increased computation time, but decrease the chance that real paths may be missed. In general, this value can be set smaller when N is larger.

The variables $R(\theta)$ and $T(\theta)$ are lookup tables for the gain of a given reflection/transmission, and are discussed further in Section 6.1.3. A complete list of variables are given in Table 6.1.

6.1.3 Wall reflection and transmission profile

In order to evaluate the effect of a wall interaction on the amplitude/gain of a particular ray, it is necessary to have some function relating the interaction coefficient with the angle for any paricular wall. This can be done using the theory discussed in Section 5.3.6. It is useful to precalculate this function for a useful range of angles as a lookup table. These lookup tables are denoted $R(\theta)$ and $T(\theta)$. It is assumed here that every environment wall has the same profile, however, if a building contains distinct wall types (for example, both glass and painted drywall) lookup tables can be created for each wall type. Also note that $R(\theta)$ and $T(\theta)$ should be defined in terms of path amplitude, not path power, a detail which is easy to overlook.

Algorithm 6.1 Ray-tracing algorithm - propagate signal vectors

Input:

Wall vectors $\{\boldsymbol{u}_0, \boldsymbol{u}_1\}_{w=1}^{W}$,
Transmitter location $\boldsymbol{l}_{\text{tx}}$,
Number of initial vectors N,
Number of ray propagation steps P
Reflection/Transmission tables $R(\theta)$ and $T(\theta)$

Output:

Propagated ray parameters, $\Theta_n^{[p]} = \{\boldsymbol{x}, \boldsymbol{v}, g, \omega, t_{\max}\}$

Begin

Create N_0 initial rays with $\boldsymbol{x}^{[0]} = \boldsymbol{l}_{\text{tx}}$, $\boldsymbol{v}^{[0]} = [\cos\frac{2\pi n}{N_0} \sin\frac{2\pi n}{N_0}]$, $p_0 = 0$, and $g^{[0]} = 1$
For $p = 0 \ldots P-1$
For $n = 0 \ldots N-1$
For $w = 1 \ldots W$
Calculate t, s for ray n and wall w using (6.2) and (6.1)
If $0 < t$ and $0 \leq s \leq 1$, **then** record collision distance t
End
Find smallest t among valid interactions, store $t_{\max,n}^{[p]} = t$, $\omega_n^{[p]} = w$
Calculate collision angle for this wall using (6.3)

Update ray segment $\boldsymbol{x}_n^{[p+1]} = \boldsymbol{x}_n^{[p]} + \boldsymbol{v}_n^{[p]} t_{\max,n}^{[p]}$
Update transmitted ray direction $\boldsymbol{v}_n^{[p+1]} = \boldsymbol{v}_n^{[p]}$
Update transmitted ray gain $g_n^{[p+1]} = T(\theta) g_n^{[p]}$

Create reflected ray and clone transmitted ray $\Theta_{\widetilde{n}}^{[0:p]} = \Theta_n^{[0:p]}$
Update reflected ray direction using (6.4)
Update reflected ray gain $g_{\widetilde{n}}^{[p+1]} = R(\theta) g_{\widetilde{n}}^{[p]}$
Update reflected ray sequence $\omega_{\widetilde{n}}^{[p]} = -w$
End
Update vector count $N = 2 \cdot N$
End

End

Table 6.1: Ray-tracing algorithm variable listing

Variable	Dimensions	Description
$p_{0,n}$	$N \times 1$	Ray start index
$\boldsymbol{x}_n^{[p]}$	$2 \times N \times P$	Ray vertex location
$\boldsymbol{v}_n^{[p]}$	$2 \times N \times P$	Ray propagation direction
$t_{\max,n}^{[p]}$	$N \times P$	Ray index of wall interaction
$g_n^{[p]}$	$N \times P$	Ray gain from interactions
$\omega_n^{[p]}$	$N \times P$	Index of wall interacted with
$\boldsymbol{u}_0, \boldsymbol{u}_1$	$2 \times W$	Start and end point for wall segments
$R(\theta)$	$L \times 1$	Lookup table for reflection coefficient
$T(\theta)$	$L \times 1$	Lookup table for transmission coefficient

6.2 Ray propagation

Rays are propagated over a number propagation steps. In each step, the rays advance according to their individual ray direction vector $\boldsymbol{v}$. These paths will continue to be propagated forward until they intersect with a wall, at which point they will split into multiple rays representing the reflections and transmissions. These new rays will then be propagated, and this process is repeated for a number of iterations (denoted here as P), or alternatively, until some stopping criterion is met.

6.2.1 Initialize rays

Before propagation, the rays need to be initialized. These rays are initialized with their first vertex as the location of the transmitter, $\boldsymbol{x}^{[0]} = \boldsymbol{l}_{\text{tx}}$. The transmission directions are then distributed radially, or $\boldsymbol{x}_n^{[0]} = [\cos \frac{2\pi n}{N_0} \;\; \sin \frac{2\pi n}{N_0}]^\top$. Gain values $g_n^{[0]}$ are set to one, or some value representing antenna transmission gain. Also, p_0 is set to zero for initial rays.

6.2.2 Identifiying next wall interaction

For each live path, the ray is checked against the list of wall segments. First, the collision point of the ray and the line made by the wall is identified. After this is done, this collision point is checked to verify that the intersection occurs in the forward portion of the ray ($0 < t$), and along the wall segment ($0 \leq s \leq 1$). The collision point can be found by

equating the ray and vectors and solving for t and s. Letting $\boldsymbol{r} = \boldsymbol{u}_1 - \boldsymbol{u}_0$, (see Appendix A.1) this leads to

$$s = \frac{\boldsymbol{v}_\perp^\top(\boldsymbol{x} - \boldsymbol{u}_0)}{\boldsymbol{v}_\perp^\top \boldsymbol{r}} \tag{6.1}$$

$$t = -\frac{\boldsymbol{r}_\perp^\top(\boldsymbol{x} - \boldsymbol{u}_0)}{\boldsymbol{v}_\perp^\top \boldsymbol{r}}, \tag{6.2}$$

where $\boldsymbol{v}_\perp = [v_2 \ -v_1]^\top$ and $\boldsymbol{r}_\perp = [r_2 \ -r_1]^\top$, which are the vectors $\boldsymbol{v}$ and $\boldsymbol{r}$ rotated 90° to the right. Further discussion is found in Appendix A.1. At this point if $t < 0$, $s > 1$, or $s < 0$ then the intersection point does not fall on the wall segment, or is behind the ray, indicating that the ray does not interact with the tested wall. Otherwise, the value t is recorded for that ray. After all wall comparisons have been made, the interaction with the smallest (nonnegative) t is chosen and saved as $t_{\max,n}$. As this interaction occurs first, all other interactions for this ray are discarded.

6.2.3 Creating transmitted and reflected rays

Once the first wall collision is identified, the collision angle is calculated using

$$\theta = \cos^{-1}\left(\frac{\boldsymbol{v}^\top \boldsymbol{r}_\perp}{\|\boldsymbol{v}\|\|\boldsymbol{r}_\perp\|}\right). \tag{6.3}$$

This will be used to calculate the gains for the reflected and transmitted paths. At this point it is necessary to consider both the transmitted ray and reflected ray, which will be treated differently. The transmitted ray will just be stored as a continuation of the original ray, while a new ray will be created to accomodate the reflected ray. The previous states of the original ray will be duplicated for the new ray, so $\Theta_{\text{reflected}}^{[0:p]} = \Theta_{\text{transmitted}}^{[0:p]}$, for all of $\boldsymbol{x}$, $\boldsymbol{v}$, g, ω, and $t_{\max}$. Since this will began at the next propagation step, $p_{0,n}$ should be set to the current propagation step value plus one, or $p+1$. The next vertex of the original path $\boldsymbol{x}_n^{[p+1]}$ is placed at the collision point $\boldsymbol{x} + \boldsymbol{v}t_{\max}$. The remaining parameters differ somewhat between transmitted and received vectors.

Transmitted vectors

The propagation direction for the transmitted vector remains unchanged, so $\boldsymbol{v}_n^{[p+1]} = \boldsymbol{v}_n^{[p]}$. The next gain $g^{[p+1]}$ is obtained using the interaction angle θ as $g^{[p+1]} = T(\theta)g^{[p]}$. The wall interaction sequence is updated as $\omega_n^{[p+1]} = w$, where w is an index identifying the wall with which the interaction occured.

Reflected vectors

The propagation direction for the next step of the reflected vector $\boldsymbol{v}_n^{[p+1]}$ is given by

$$\boldsymbol{v}_{\text{reflected}} = \boldsymbol{v} - 2(\text{proj}_{\boldsymbol{r}_\perp}\boldsymbol{v}) = \boldsymbol{v} - \frac{2\boldsymbol{v}^\top \boldsymbol{r}_\perp}{||\boldsymbol{r}_\perp||^2}\boldsymbol{r}_\perp. \tag{6.4}$$

The next reflected vector gain $g^{[p+1]}$, like the transmitted vector, is found using the interaction angle θ as $g^{[p+1]} = R(\theta)g^{[p]}$. Note that $R(\theta)$ typically contains a sign flip, which represents a 180° phase change often caused in a reflection. The wall interaction sequence is updated almost identically to the transmitted vector as $\omega_n^{[p+1]} = -w$, with the only difference as using a negative value to indicate a reflected path. Note that this is an optional shortcut, it is simply a convenient way to store if an event was a transmission or reflection. This also relies on wall indices starting at one. Depending on implementation details, it may be easier to store this as a separate boolean.

6.3 Identify received paths

After all paths out from the transmitter are calculated, one can use these to determine which paths connect to the receiver. The algorithm proceeds by looping over each ray at each propagation step, over which the ray takes the form of a line segment. Only rays for which the starting step p_0 is greater than the currently checked step p are investigated. This prevents multiple rays which follow the same initial path from being processed additional times.

In each propagation step, all ray segments are compared to the location of each receiver, and if the segment passes close enough, it is recorded. An additional step is then performed

to eliminate duplicate paths which follow the same reflection/transmission sequence. Once this is done, the distance and power of each path reaching a particular receiver will be recorded.

Algorithm 6.2 Ray-tracing algorithm - compute received paths (one receiver)

Input:
Receiver location(s) $\{\boldsymbol{l}_{\text{rx}}\}$,
Propagated ray parameters, $\Theta_n^{[p]} = \{\boldsymbol{x}, \boldsymbol{v}, g, \omega, t_{\max}\}$
Output:
Received path distances, $\{d\}$
Received path gains, $\{\alpha\}$
Begin
For $p = 0 \ldots P-1$
For $n = 0 \ldots N-1$
Find closest point on ray using (6.5), then $\boldsymbol{x}_{\text{closest}} = \boldsymbol{x} + \boldsymbol{v}t$
Calculate ray-point distance $d_{\text{line-to-rx}}$ using (6.6)
If $0 < t \leq t_{\max}$ and $d_{\text{line-to-rx}} \leq d_{\text{rx}}$, **then** add p to list of possible receives
End
if $\omega_i^{[p]} = \omega_j^{[p]}$ for any $i \neq j$ in possible receive list, **then**
Discard receive with higher $d_{\text{line-to-rx}}$
Record received path distance d from (6.7) and gain α from (6.8)
End
End

6.3.1 Find receiver intersections

To check if a path crosses a receiver, the closest position on the line segment is found by first identifying a position index t using

$$t = \frac{\boldsymbol{v}^\top(\boldsymbol{l}_{\text{rx}} - \boldsymbol{x})}{\|\boldsymbol{v}\|}. \tag{6.5}$$

Then the closest point on the line segment can be identified as $\boldsymbol{x}_{\text{closest}} = \boldsymbol{x} + \boldsymbol{v}t$. This closest point is checked to verify it does not extend past the edge of the line segment. If $0 \leq t \leq t_{\max}$, then the point is within the bounds of the ray segment, and the point-line

distance is checked. Otherwise (if $t < 0$ or $t > t_{\max}$), the point does not fall on the line segment, and no interaction is recorded. If the closest point lies on the ray segment, the point-line distance is calculated as

$$d_{\text{line-to-rx}} = \|\boldsymbol{l}_{\text{rx}} - (\boldsymbol{x} + \boldsymbol{v}t)\|. \tag{6.6}$$

If the receiver falls within a previously determined distance d_{rx} of the ray, it is recorded as a receive.

6.3.2 Removing duplicate paths

In practice, multiple rays following the what is essentially the same path may fall within the radius set for the receiver. For each path received at a receiver, the sequence of walls that that ray interacted with up to that point $\omega^{[0:p-1]}$ is compared to the other received paths. Note that two paths i and j are only considered equivalent if $\omega_i^{[k]} = \omega_j^{[k]}$ for every element $k \in (0 \dots p-1)$. If a set of paths have the same wall sequence, that suggests that they should be the same path (at least as observed by this receiver). The $d_{\text{line-to-rx}}$ value for each of these paths is then compared, and the path with the minimum value is kept, and other equivalent paths are discarded. After all duplicate paths have been removed, the gain and delay for each received path is computed.

6.3.3 Calculating received path gain and time delay

After accounting for matching paths, the distance travelled by the ray until it reaches the point on the line-segment which the receiver projects onto is recorded as a receiver path distance. This is found by summing the lengths of distances up to the current propagation step, then adding the distance from the last vertex to the receiver location as

$$d = \left(\sum_{k=0}^{p-1} \|\boldsymbol{x}_n^{[k+1]} - \boldsymbol{x}_n^{[k]}\| \right) + \|\boldsymbol{x}_n^{[p]} - \boldsymbol{l}_{\text{rx}}\|. \tag{6.7}$$

The received path gain is found by combining the path loss with the gain calculated from wall interactions. This can be expressed as

$$\alpha = \frac{1}{d^{(\gamma/2)}} \cdot g_n^{[p-1]}, \tag{6.8}$$

where γ is the loss coefficient discussed in Section 5.3.5, but typically has a value of 2. This formula assumes that distances are in meters. The final output of the receive process is the set of every registered received path distance and gain values $\{\boldsymbol{d}, \boldsymbol{\alpha}\}$ for each receiver.

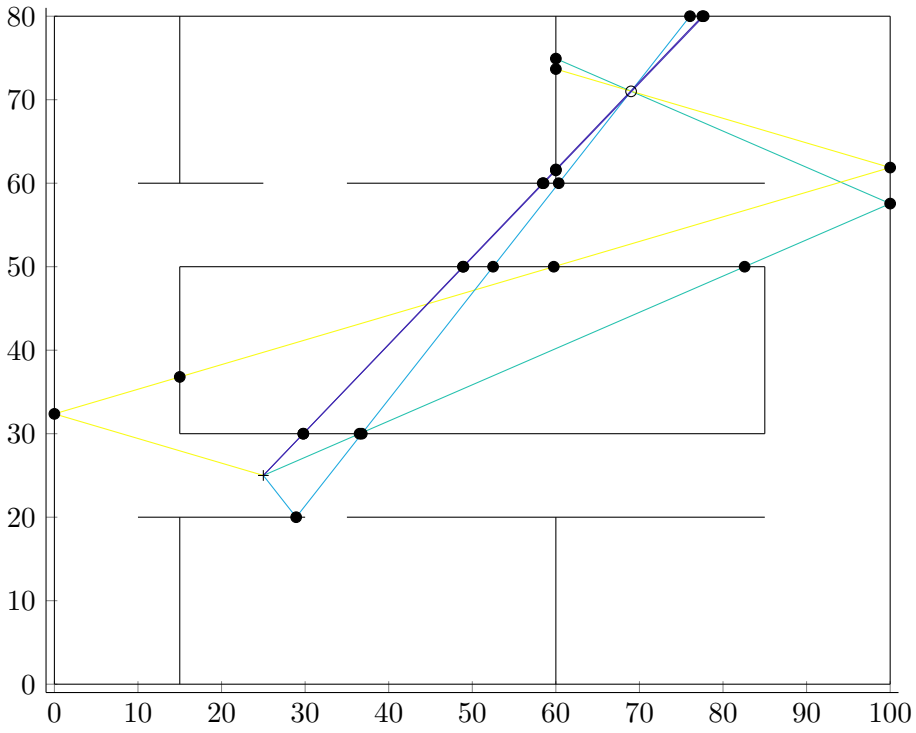


Figure 6.1: A sample of ray-tracing results. Darker paths indicate higher received power.

6.3.4 Generating an impulse response function

Once ray tracing is completed, the continuous channel between the transmitter and a receiver k is described by the impulse response:

$$h(t) = \sum_{i=0}^{N-1} \alpha_i \delta(t - \tau_i), \tag{6.9}$$

where $d = \frac{d}{c}$ (with c as the speed of light in meters per second).

For simulating a discrete system, this must be adjusted as described in Section 5.1 and Section 5.2. The discrete system channel impulse response is given by (5.12), which incorporates the various of effects of anti-aliasing, sampling and mixing.

6.3.5 Other considerations

Some ray-tracing algorithms (like that in [36]) model rays with a cone structure rather than a particular radius. Since the reflection and transmission coefficients are different for different polarizations, if multiple polarizations are considered these must be treated separately, though paths will be not be changed, so gains associated with each polarization can be calculated simultaneously without substantial additional cost.

Ray-tracing can be performed in two or three dimensions, with an increase in accuracy and computational cost in the three-dimensional case. This algorithm is described in two dimensions. Extending this algorithms to three dimensions is relatively straightforward (rays become three-dimensional and walls become planes), though computational cost and polarization considerations may require more attention.

6.3.6 Reversibility

It is also worth noting that receivers and transmitters can often be interchanged. This follows from the principle of reversibility, which states that when there is no absorption in materials, radiated waves will take the same path moving between two points regardless of direction (see Hecht [44]). It is also clear that since entry and exit angles for walls are the same from both directions, the gain for the path in each direction will also be equivalent.

This allows the results for the multipath between two points to be interpreted with either endpoint as either the transmitter or receiver.

CHAPTER 7

Conclusion and Final Femarks

7.1 Project contributions and discussion

In this thesis, a Gaussian-process-based method for autocorrelation based fingerprinting have been proposed, along with analysis of its performance. In this thesis, the possibility of RF source localization by utilizing signal autocorrelations to perform fingerprinting was explored. A Gaussian-process-based method was proposed, including a strategy to estimate model parameters efficiently. The performance of this method was then analyzed using a ray-tracing simulation environment.

7.1.1 Performance and applicability of autocorrelation-based fingerprinting

Ultimately, this research demonstrates that, at least in simulated scenarios, autocorrelation-based fingerprinting can be used for performing RF-source localization. The results here (such as those depicted in Section 4.10) still suggest that a significant amount of bandwidth (on the order of 8MHz or higher) is really necessary to get reasonable performance, and even then multiple receivers are likely necessary. Still, the localization capability may still be nonnegligible even in the lower end of these cases, so this information may be useful when included in other power, phase, or timing based solutions. Autocorrelation fingerprinting has the advantage of possibly being added to many simple, unsynchronized software-defined radio localization systems without any additional hardware. This is perhaps the application of this research which is most likely to be useful, especially since received-signal-strength systems can already effectively utilize a Gaussian procees model, as described in Aravecchia and Messelodi [29] and Ferris et al. [25]. Of course fingerprinting will still require the data collected from the environment-of-interest, but if other fingerprinting on RSS is already being performed, this may not add much additional work.

The methods described in this thesis fall in a broader category of methods which rely on channel information to estimate the position of a noncooperative transmitter. While the autocorrelation has the advantage of being leverageable without much additional information, in practice it may be preferable to make try to estimate and leverage signal structure, such as pulse-shape (which clearly has an effect as seen in Figure 4.14). The Gaussian process method could also be used on other forms of channel information such as channel state information, as done by Zhao et al. in [15].

7.1.2 The Gaussian process model for fingerprinting

The Gaussian process model used seems reasonably functional, and has many advantages over a more naive approach, but may still be far from the best model for this particular application. Other developed methods, such as that developed by Hall in [6] and [11], may already outperform this method, though comparison of these methods was unfortunately beyond what was possible with this thesis. A comparison of the methods in this work, those from Hall et al., and any other similar work might help guide future efforts to develop fingerprinting methods.

It is also quite possible that a more capable variant of the Gaussian random process model used here could be produced by changing some of its assumptions or construction. The particular variance kernel could have been modified. Additionally, the physical distance d could have been replaced by a distance which reflects more information about the environment, such as by artificially increasing d for points which were not in the same room or otherwise separated by an obstruction. An example of this might be a d value based on the shortest non-obstructed floor path between two points, as if "walking" between the two points was required by the transmission. The kernel used for the covariance could also be changed from the Gaussian kernel, as this was chosen based on convenience and precedent rather than chosen to suit the scenario. Alternatively, perhaps another form of random process would better model some of the less smooth effects of position on autocorrelation and its features.

7.1.3 Gaussian process parameter estimation

The least squares parameter estimation method described in Section 3.2.4 adequately met the needs of this research for quick and reliable Gaussian process parameter estimation. It is possible that this method could be useful in other similar applications of Gaussian processes, or as a method of producing initial values for a maximum-likelihood descent method. Alternatively, it is possible also that better methods already exist for this step, and these methods could be used to perform localization performance in this application. There were also instances where the estimated parameters for the Gaussian process had a lower likelihood than a "no information" parameter set obtainable using true maximum likelihood. This suggests that performance may be improvable by more optimal parameter selection, though it is the opinion of the author that these improvements would likely be fairly modest.

7.2 Other areas of further work

While reasonable effort was invested in making an effective simulation, the simulation still ignored many real-world factors, such as interference from other signals, three-dimensional environments, mobile and immobile reflective objects, and signal diffraction. It would be useful to perform analysis and testing which accounts for all of these. It would also be preferrable to perform testing with data collected using real hardware.

7.3 Final remarks

The primary success of this research is likely the evidence that autocorrelation-based fingerprinting may offer real information about transmitter location in the particular scenarios it can be used. It does not seem like in most cases this is an optimal approach, but may still be a helpful tool in the niche situations where a noncooperative transmitter must be located in a controlled environment.

REFERENCES

[1] H. C. So, *Source Localization: Algorithms and Analysis*. IEEE, 2012, pp. 25–66. [Online]. Available: https://ieeexplore.ieee.org/document/6047813

[2] X. Li, Z. D. Deng, L. T. Rauchenstein, and T. J. Carlson, "Contributed review: Source-localization algorithms and applications using time of arrival and time difference of arrival measurements," *Review of Scientific Instruments*, vol. 87, no. 4, p. 041502, 2016. [Online]. Available: https://doi.org/10.1063/1.4947001

[3] S. Whiting, T. K. Moon, and J. H. Gunther, "Geolocation from received signal strength," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, 10 2018, pp. 1147–1151.

[4] R. K. Martin, A. S. King, J. R. Pennington, R. W. Thomas, R. Lenahan, and C. Lawyer, "Modeling and mitigating noise and nuisance parameters in received signal strength positioning," *IEEE Transactions on Signal Processing*, vol. 60, no. 10, pp. 5451–5463, Oct 2012.

[5] C. Comsa, "Source localization via time difference of arrival," Ph.D. dissertation, New Jersey Institute of Technology, 01 2012.

[6] D. L. Hall, R. M. Narayanan, E. H. Lenzing, and D. M. Jenkins, "Passive vector sensing for non-cooperative emitter localization in indoor environments," *Electronics*, vol. 7, no. 12, 2018. [Online]. Available: https://www.mdpi.com/2079-9292/7/12/442

[7] K. Pahlavan, Xinrong Li, and J. P. Makela, "Indoor geolocation science and technology," *IEEE Communications Magazine*, vol. 40, no. 2, pp. 112–118, Feb 2002.

[8] C. S. Agate, M. Varble, and K. O. Ezal, "Ground-based emitter location in the presence of multipath," in *2019 IEEE Aerospace Conference*, March 2019, pp. 1–8.

[9] R. Saeed, S. Khatun, B. Ali, and M. Khazani, "Ultra-wideband (UWB) geolocation in NLOS multipath fading environments," in *2005 13th IEEE International Conference on Networks Jointly held with the 2005 IEEE 7th Malaysia International Conf on Communic*, vol. 2, Nov 2005, pp. 6 pp.–.

[10] F. Yin, C. Fritsche, F. Gustafsson, and A. M. Zoubir, "Received signal strength-based joint parameter estimation algorithm for robust geolocation in LOS/NLOS environments," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 6471–6475.

[11] D. L. Hall, R. M. Narayanan, and D. M. Jenkins, "SDR based indoor beacon localization using 3D probabilistic multipath exploitation and deep learning," *Electronics*, vol. 8, no. 11, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/11/1323

[12] A. O'connor, P. Setlur, and N. Devroye, "Single-sensor RF emitter localization based on multipath exploitation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 51, no. 3, pp. 1635–1651, July 2015.

[13] Lang Tong and S. Perreau, "Multichannel blind identification: from subspace to maximum likelihood methods," *Proceedings of the IEEE*, vol. 86, no. 10, pp. 1951–1968, Oct 1998.

[14] D. Darsena, G. Gelli, L. Paura, and F. Verde, "Subspace-based blind channel identification of siso-fir systems with improper random inputs," *Signal Processing*, vol. 84, pp. 2021–2039, 11 2004.

[15] L. Zhao, H. Wang, P. Li, and J. Liu, "An improved wifi indoor localization method combining channel state information and received signal strength," in *2017 36th Chinese Control Conference (CCC)*, July 2017, pp. 8964–8969.

[16] M. L. Rose, J. L. Ipson, T. K. Moon, and J. H. Gunther, "Sparse adaptive channel identification method using the cross-relation," *Physical Communication*, vol. 47, p. 101354, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874490721000914

[17] C. Nerguizian, C. Despins, S. Affes, G. I. Wassi, and D. Grenier, "Neural network and fingerprinting-based geolocation on time-varying channels," in *2006 IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications*, Sep. 2006, pp. 1–6.

[18] B. Baykal, "Blind channel estimation via combining autocorrelation and blind phase estimation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 51, no. 6, pp. 1125–1131, June 2004.

[19] K. Jaganathan, Y. C. Eldar, and B. Hassibi, "Phase retrieval: An overview of recent developments," *ArXiv*, vol. abs/1510.07713, 2015.

[20] Y. Qi, H. Kobayashi, and H. Suda, "Analysis of wireless geolocation in a non-line-of-sight environment," *IEEE Transactions on Wireless Communications*, vol. 5, no. 3, pp. 672–681, 2006.

[21] F. O. Akgul and K. Pahlavan, "A model for spatial behavior of multipath components pertinent to indoor geolocation using ray optics," *International Journal of Wireless Information Networks*, vol. 20, no. 4, pp. 328–345, Dec 2013. [Online]. Available: https://doi.org/10.1007/s10776-013-0225-5

[22] C. Nerguizian, C. Despins, and S. Affès, "Indoor geolocation with received signal strength fingerprinting technique and neural networks," in *Telecommunications and Networking - ICT 2004*, J. N. de Souza, P. Dini, and P. Lorenz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 866–875.

[23] X. Wang, L. Gao, S. Mao, and S. Pandey, "CSI-based fingerprinting for indoor localization: A deep learning approach," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 1, pp. 763–776, Jan 2017.

[24] M. N. de Sousa and R. Thom, "Enhancement of localization systems in nlos urban scenario with multipath ray tracing fingerprints and machine learning," *Sensors*, vol. 18, p. 4073, 11 2018.

[25] B. Ferris, D. Hhnel, and D. Fox, "Gaussian processes for signal strength-based location estimation," in *In Proc. of Robotics Science and Systems*, 08 2006.

[26] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[27] T. K. Moon and W. C. Stirling, *Mathematical Methods and Algorithms for Signal Processing*. Upper Saddle River, NJ: Prentice Hall, 2000, ch. 13, pp. 591–620.

[28] J. L. Ipson and T. K. Moon, "Source localization on limited bandwidth signals using autocorrelation-based fingerprinting," in *2021 55th Asilomar Conference on Signals, Systems, and Computers*, 2021, pp. 528–533.

[29] M. Aravecchia and S. Messelodi, "Gaussian process for RSS-based localisation," in *2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2014, pp. 654–659.

[30] C. Nerguizian and V. Nerguizian, "Indoor fingerprinting geolocation using wavelet-based features extracted from the channel impulse response in conjunction with an artificial neural network," in *2007 IEEE International Symposium on Industrial Electronics*, June 2007, pp. 2028–2032.

[31] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. Sebastopol, CA: O'Reilly Media, 2017.

[32] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, Massachusetts and London, England: The MIT Press, 2006. [Online]. Available: http://www.gaussianprocess.org/gpml/

[33] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil, "Fast direct methods for gaussian processes," 2014. [Online]. Available: https://arxiv.org/abs/1403.6015

[34] S. Das, S. Roy, and R. Sambasivan, "Fast gaussian process regression for big data," 2015. [Online]. Available: https://arxiv.org/abs/1509.05142

[35] F. Wang, H. Li, W. Xia, and J. Zhou, "The maximum of CRLB of time delay estimation," in *2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS)*, Dec 2014, pp. 1–4.

[36] A. H. L. Kaveh Pahlavan, *Wireless Information Networks (Wiley Series in Telecommunications and Signal Processing)*. River Street, Hoboken NJ, United States: John Wiley & Sons, Inc., 2005, ch. 6.

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

[38] A. Regmi, "Reflection measurement of building materials at microwaves," Ph.D. dissertation, 01 2016.

[39] A. Navarro, D. Guevara, J. Gimenez, and N. Cardona, "Measurement-based ray-tracing models calibration of the permittivity and conductivity in indoor environments," *Ingeniera y Competitividad*, vol. 20, pp. 41–51, 06 2018.

[40] C. A. Balanis, *Advanced Engineering Electromagnetics*. New York: John Wiley & Sons, Inc., 1989, ch. 5, pp. 180–253.

[41] F. A. Alves, M. R. M. L. de Albuquerque, S. G. da Silva, and A. G. d'Assuncao, "Efficient ray-tracing method for indoor propagation prediction," in *SBMO/IEEE MTT-S International Conference on Microwave and Optoelectronics, 2005.*, July 2005, pp. 435–438.

[42] S. Hosseinzadeh, "3D ray tracing for indoor radio propagation," https://www.mathworks.com/matlabcentral/fileexchange/64695-3d-ray-tracing-for-indoor-radio-propagation, May 2019, MATLAB Central File Exchange. Retrieved April 19, 2022.

[43] T. K. Geok, F. Hossain, and A. T. W. Chiat, "A novel 3D ray launching technique for radio propagation prediction in indoor environments," *PLOS ONE*, vol. 13, no. 8, pp. 1–14, 08 2018. [Online]. Available: https://doi.org/10.1371/journal.pone.0201905

[44] E. Hecht, *Optics*. Pearson, 2012.

[45] D. G. W. Gwilym M. Jenkins, *Spectral Analysis and Its Applications*. Holden-Day, 1968.

[46] A. Seijas-Macías, A. Oliveira, T. A. Oliveira, and V. Leiva, "Approximating the distribution of the product of two normally distributed random variables," *Symmetry*, vol. 12, p. 1201, 2020.

[47] N. Schlmer, "matlab2tikz," https://www.mathworks.com/matlabcentral/fileexchange/22022-matlab2tikz-matlab2tikz, Aug. 2022, MATLAB Central File Exchange. Retrieved October 18, 2022.

APPENDICES

APPENDIX A

Additional Content

A.1 Ray intersection points

This section describes the basic geometry governing the ray-wall intersections, ray reflections, and point-ray interations relevant to the ray-tracing algorithm described in Chapter 6.

It should be noted that there are multiple possible mathematical representations of rays and line segments, and this can effect the form of these calculations.

Certain forms may be more or less convenient depending on how data is handled, and whether the simulation has two or three dimensions.

Ray-wall Intersections

With the ray described as $\{\boldsymbol{x}+\boldsymbol{v}t|t>0\}$ and the wall described as $\{\boldsymbol{u}_0(s)+\boldsymbol{u}_1(1-s)\}$, as given in Section 6.1.2, let $\boldsymbol{r}=\boldsymbol{u}_1-\boldsymbol{u}_0$. One can then find the point in both of these sets

(the intersection point) by equating them:

$$\boldsymbol{x} + \boldsymbol{v}t = \boldsymbol{u}_0 + \boldsymbol{r}s \tag{A.1}$$

$$\implies \boldsymbol{x} - \boldsymbol{u}_0 = \boldsymbol{r}s - \boldsymbol{v}t = \begin{bmatrix} \boldsymbol{r} & \boldsymbol{v} \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} \tag{A.2}$$

$$\implies \begin{bmatrix} s \\ t \end{bmatrix} = \begin{bmatrix} \boldsymbol{r} & \boldsymbol{v} \end{bmatrix}^{-1} (\boldsymbol{x} - \boldsymbol{u}_0) \tag{A.3}$$

$$= (\frac{1}{\boldsymbol{v}_\perp^\top \boldsymbol{r}}) \begin{bmatrix} \boldsymbol{v}_\perp^\top \\ -\boldsymbol{r}_\perp^\top \end{bmatrix} (\boldsymbol{x} - \boldsymbol{u}_0) \tag{A.4}$$

$$\implies s = \boldsymbol{v}_\perp^\top (\boldsymbol{x} - \boldsymbol{u}_0)/(\boldsymbol{v}_\perp^\top \boldsymbol{r}) \tag{A.5}$$

$$\implies t = -\boldsymbol{r}_\perp^\top (\boldsymbol{x} - \boldsymbol{u}_0)/(\boldsymbol{v}_\perp^\top \boldsymbol{r}). \tag{A.6}$$

This results in the equations given in Section 6.2.2.

Ray Reflections

As shown in Figure A.1, the reflected direction vector is unchanged in the axis parallel to the wall surface. The only change is that the vector is inverted along the axis perpendicular to the surface. This can be achieved by substracting twice that projection from the original vector. This gives (6.4).

It can also be seen that the angle of incidence is simply the angle between $\boldsymbol{v}$ and $\boldsymbol{r}_\perp$, which is found using the standard formula for the angle between two vectors, resulting in (6.3).

A.2 Simulating nonideal autocorrelation estimates

To produce properly simulated autocorrelation, it is necessary to account for the effects of estimating the statistical signal autocorrelation from a limited, noisy sample. One method for this is to generate a test signal $s_{\text{baseband}}(n)$, convolve it with the channel impulse response $h_{\text{sys}}(n)$, add the noise $w(n)$, and calculate the autocorrelation as in (5.13). However, if

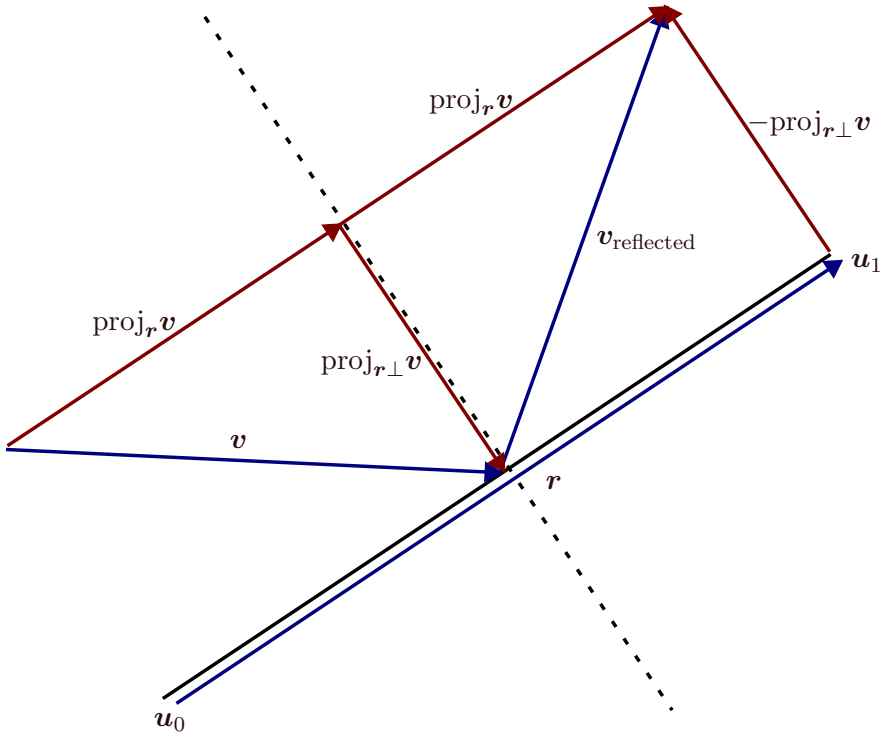


Figure A.1: The vector for a reflected ray $\boldsymbol{v}_{\text{reflected}}$ is the vector for the initial ray $\boldsymbol{v}$, except with the sign of the component in the perpendicular wall direction $\boldsymbol{r}_\perp$ inverted.

the number of data points N used becomes long, this can begin to be computationally expensive. If a large number of simulated autocorrelations is needed this complexity can become prohibitive.

As an alternative that can produce nearly equivalent results at a practical computational cost, the simulated autocorrelation was instead produced by simulating the individual autocorrelation components in found in (5.16), repeated here for convenience:

$$r_y(k) = [e^{j2\pi\Delta f k T} r_{\text{sys}}(k)] * r_{\text{sig}}(k) + [r_{\text{cross}}(k) + r_{\text{cross}}{}^*(-k)] + r_{\text{w}}(k). \tag{A.7}$$

For this process $r_{\text{sys}}(k)$ is deterministic and can be calculated directly from $h_{\text{sys}}(k)$. That leaves the other three terms $r_{\text{sig}}(k)$, $r_{\text{cross}}(k)$, and $r_{\text{w}}(k)$, which will need to be generated.

If $s_{\text{bb}}(n)$ and $w(n)$ are modeled as random processes, then it follows that these terms will be random processes with a distribution relating to the autocorrelation estimation process. Fortunately, since each term in one of these autocorrelation processes is the sum

of many product terms which are themselves random, the central limit theorem can be invoked. It is therefore justifiable to model $r_{\text{sig}}(k)$, $r_{\text{cross}}(k)$, and $r_{\text{w}}(k)$ as Gaussian random processes. These processes are assumed to be independent for simplicity ($r_{\text{sig}}(k)$ and $r_{\text{w}}(k)$ are reasonably assumed to be independent, but $r_{\text{cross}}(k)$ has some entanglement with both $r_{\text{sig}}(k)$ and $r_{\text{w}}(k)$, which is ignored).

Gaussian random processes can be easily simulated if the mean and covariance is known. It is assumed that $s_{\text{baseband}}(k)$ has a power of one (and therefore unity variance), and adjust signal noise power N_0^2 to achieve the desired signal-to-noise ratio. Note that in Chapter 4, signal-to-noise ratio is set according to the median received power, which requires calculating the power associated with every channel $r_{\text{cross}}(k)$ and assigning N_0^2 relative to that value.

An approximation for the mean and covariance of autocovariance estimates are given in Jenkins and Watts [45]. Letting $r_{yy}(n) = \frac{1}{N-k}\sum_{n=0}^{N-1} y(n)y(n-k)$ be the autocorrelation estimate, and assuming $E[y(n)] = 0$ (this renders the autocorrelation and autocovariance equivalent), and letting $\gamma_{yy}(k)$ be the theoretical autocorrelation obtained for $N = \infty$, these are included below, with minor adjustments to adapt them to autocorrelation estimates used in this thesis.

$$E[r_y y(k)] = \gamma_{yy}(k), \quad |k| \leq N \tag{A.8}$$

$$\text{Cov}[r_{yy}(k), r_{yy}(l)] = \frac{1}{N-k}\sum_{n=-\infty}^{\infty} \gamma_{yy}(n)\gamma_{yy}(n+l-k) + \gamma_{yy}(n+l)\gamma_{yy}(n-k). \tag{A.9}$$

If $s_{\text{bb}}(n)$ and $r_{\text{w}}(n)$ are white noise processes, they have $\gamma_{yy}(k) = \delta(k)$ and $\gamma_{yy}(k) = N_0^2\delta(k)$ respectively. The sum in (A.8) will then reduce to

$$\begin{aligned}
&\frac{1}{N-k}\sum_{n=-\infty}^{\infty} \gamma_{yy}(n)\gamma_{yy}(n+l-k) + \gamma_{yy}(n+l)\gamma_{yy}(n-k) \\
&= \frac{1}{N-k}\sum_{n=-\infty}^{\infty} N_0^4\delta(n)\delta(n+l-k) + N_0^4\delta(n+l)\delta(n-k) \\
&= \frac{N_0^4}{N-k}(\delta(l-k) + \delta(k+l)),
\end{aligned} \tag{A.10}$$

with $N_0 = 1$ for $r_{\text{sig}}(k)$. This suggests that if the $\frac{1}{N-k}$ scaling is applied later, then

the covariance will be 1 or N_0^4 for individual samples $k \neq 0$, having double that variance for the $k = 0$ term. This also indicates that pairs $k = -l$ must be identical, as $\mathrm{Cov}[r_{yy}(k), r_{yy}(l)] = \mathrm{Cov}[r_{yy}(k), r_{yy}(k)] = \mathrm{Cov}[r_{yy}(l), r_{yy}(l)]$, meaning that all variance is completely shared. This is actually a rather mundane result, as it only enforces the condition that the autocorrelation is symmetric (in the real case), which must always be true of real autocorrelation functions. Still, this gives a way to produce a sample $r_{\mathrm{sig}}(k)$ and $r_{\mathrm{w}}(k)$. It can be produced generating a random vector drawn from

$$\mathcal{N}\left(\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 2 & \mathbf{0}^\top \\ \mathbf{0} & I \end{bmatrix}\right) \tag{A.11}$$

for $r_{\mathrm{sig}}(k)$, and then (if values for $k < 0$ are needed) extending it using the autocorrelation symmetry property. Likewise samples of $r_{\mathrm{w}}(k)$ are produced using

$$\mathcal{N}\left(\begin{bmatrix} N_0^2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 2N_0^4 & \mathbf{0}^\top \\ \mathbf{0} & N_0^4 I \end{bmatrix}\right) \tag{A.12}$$

and extended for $k < 0$ using symmetry if needed.

To produce a sample for $r_{\mathrm{cross}}(k)$, slightly different methodology is needed. One can first calculate a sequence for $s_{\mathrm{bb}}(n) * w(-n)$. Each term in the sum $\frac{1}{N-k}\sum_{n=0}^{N-1} s_{\mathrm{bb}}(n)w(n-k)$ is the product of two Gaussian random variables, which are then added together. The product of two Gaussian random variables with mean zero and variance one is a nongaussian random variable, which also has variance one (this follows trivially from results by Seijas-Macías et al. in [46]). Since the component Gaussian random variables have variances of 1 and N_0^2, the product variance will be N_0^2 (as $\mathrm{Cov}[1 \cdot XN_0Y] = N_0^2\mathrm{Cov}[XY]$). Since these nongaussian-distributed terms are summed however, central limit theorem can be invoked once again

to justify using a Gaussian random process to represent these values. The $s_{\text{bb}}(n) * w(-n)$ values can then be treated as a Gaussian random process with mean zero and a covariance of N_0^2. One can then convolve this signal with $h_{\text{sys}}(n)$, giving a sequence for $r_{\text{cross}}(k)$. The sequence $s_{\text{bb}}(n) * w(-n)$ must be made long enough to avoid edge effects of $r_{\text{cross}}(k)$, and central values chosen. Finally the generated sequences will be combined as in (*A.7*), and the scaling $\frac{1}{N-k}$ is applied to whole sequence.

Note that this process neglects the role of complex phase. If it is assumed that the phase of $s_{\text{baseband}}(n)$ and $w(n)$ are uniformly random and independent between each sample, it is reasonable to conclude that the phase of the product terms in the autocorrelation will be similarly random. There is no mechanism to encourage any bias towards a particular phase in the resulting autocorrelation (with the exception of the autocorrelation conjugate symmetry), so it is assumed that the phase is random for each sample up to conjugate symmetry. One can therefore simply generate samples of $r_{\text{sig}}(k)$, $r_{\text{w}}(k)$, and $r_{\text{cross}}(k)$ with random phase for each sample (and forcing conjugate symmetry except on $r_{\text{cross}}(k)$.

Results of this process are compared to the baseline method of generating source signals $s_{\text{baseband}}(n)$ performing convolution, adding noise and calculating the autocorrelation from the whole sequence in Figures A.2 — A.5.

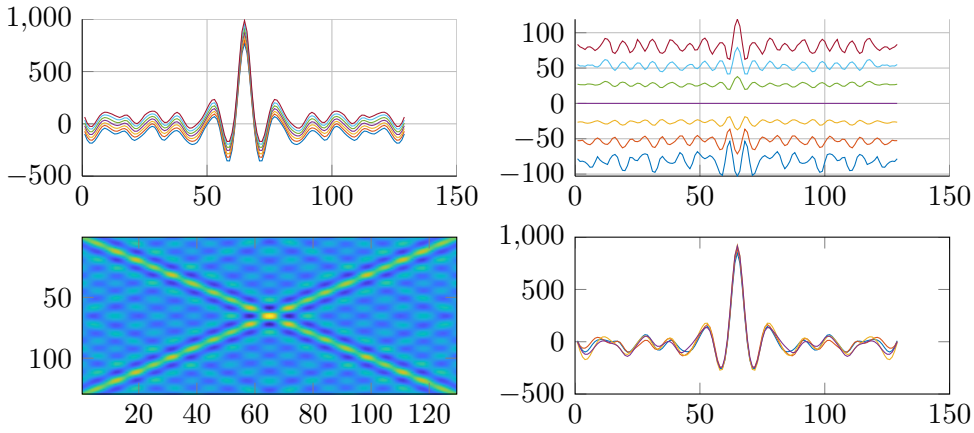


Figure A.2: Autocorrelation samples from baseline method, looking at real part of sequences. (Top left) Median autocorrelation with 0.0013, 0.228, 0.1587, 0.8413, 0.9772, and 0.9987 quantile regions, corresponding to standard deviations for Gaussian-distributed variables. (Top right) Same quantile regions as distance from median. (Bottom left) The normalized autocorrelation matrix from these samples with brighter regions indicating higher correlation. (Bottom right) Several autocorrelation sequences generated using this method. (Top left) Median autocorrelation with quantile regions

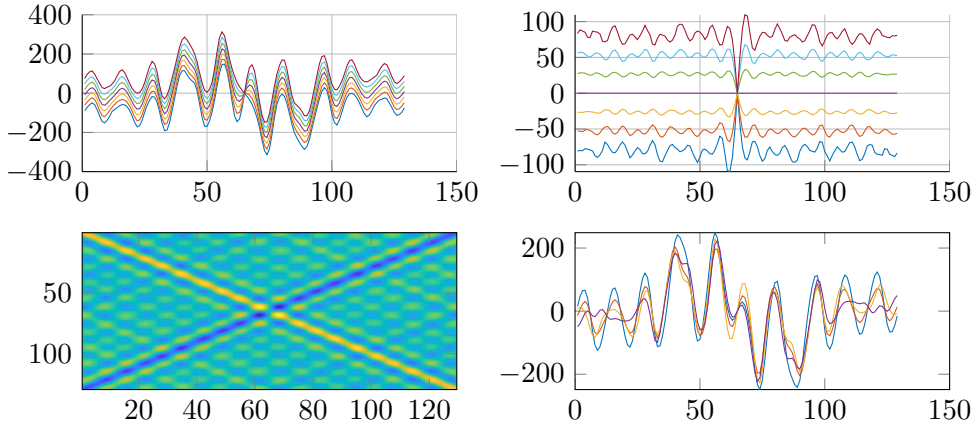


Figure A.3: Autocorrelation samples from baseline method, looking at imaginary part of sequences. (Top left) Median autocorrelation with 0.0013, 0.228, 0.1587, 0.8413, 0.9772, and 0.9987 quantile regions, corresponding to standard deviations for Gaussian-distributed variables. (Top right) Same quantile regions as distance from median. (Bottom left) The normalized autocorrelation matrix from these samples with brighter regions indicating higher correlation. (Bottom right) Several autocorrelation sequences generated using this method. (Top left) Median autocorrelation with quantile regions

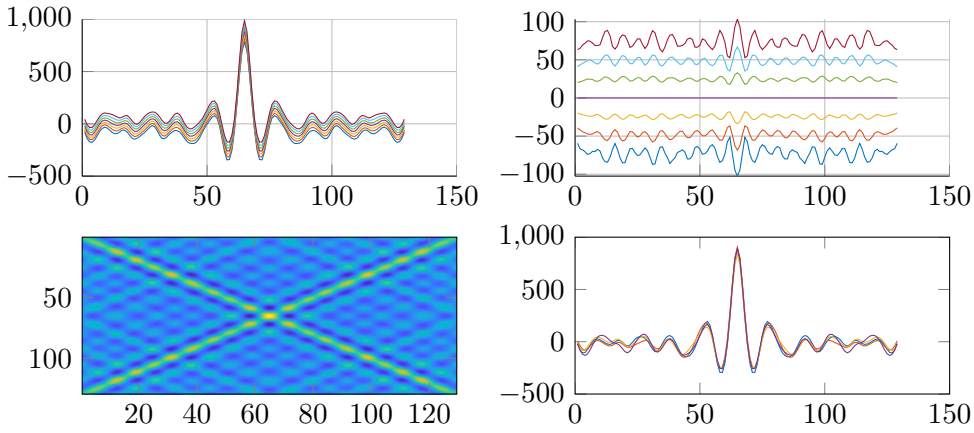


Figure A.4: Autocorrelation samples from described method, looking at real part of sequences. (Top left) Median autocorrelation with 0.0013, 0.228, 0.1587, 0.8413, 0.9772, and 0.9987 quantile regions, corresponding to standard deviations for Gaussian-distributed variables. (Top right) Same quantile regions as distance from median. (Bottom left) The normalized autocorrelation matrix from these samples with brighter regions indicating higher correlation. (Bottom right) Several autocorrelation sequences generated using this method. (Top left) Median autocorrelation with quantile regions

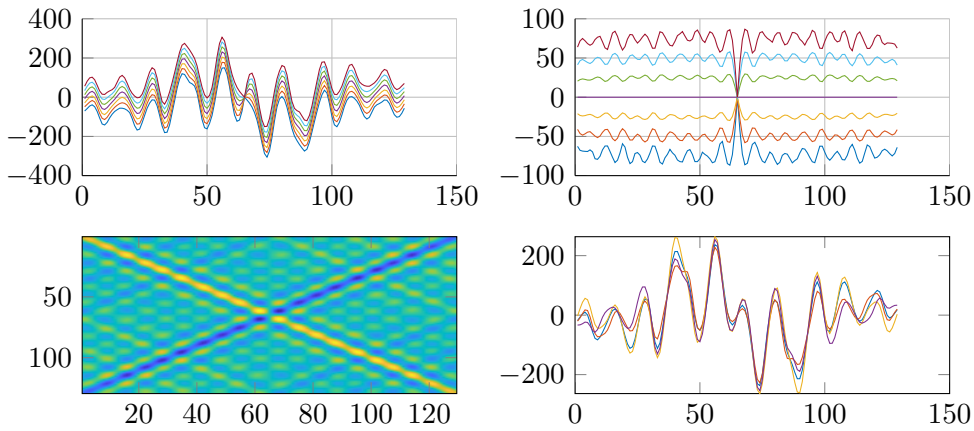


Figure A.5: Autocorrelation samples from described method, looking at imaginary part of sequences. (Top left) Median autocorrelation with 0.0013, 0.228, 0.1587, 0.8413, 0.9772, and 0.9987 quantile regions, corresponding to standard deviations for Gaussian-distributed variables. (Top right) Same quantile regions as distance from median. (Bottom left) The normalized autocorrelation matrix from these samples with brighter regions indicating higher correlation. (Bottom right) Several autocorrelation sequences generated using this method. (Top left) Median autocorrelation with quantile regions

APPENDIX B

Code Listings

B.1 Overview of source code

This section contains the basic MATLAB® code used in this project. These scripts can be used to reproduce the main results in Chapter 4. The plots used in this thesis were derived from this code, additionally using the matlab2tikz package [47].

B.2 Main scripts

The following four MATLAB® scripts can be used to perform ray-tracing, generate impulse response functions, and finally evaluate fingerprinting performance. They require file reading/writing access and MATLAB® GPU functions, though it would not be difficult to modify them to remove dependence on these GPU functions if needed.

script_genSantFloor.m

```
% script - genSantFloor
%   generates the .mat file that is used to create the WallSet
    used
%   elsewhere in this code
% requires:
%
% outputs:
%   wallSet.mat file,  containing u0,  and u1 variables
%

u0 = [...
40.5, 16.5;
40.5, 26.5;
   7, 26.5;
   7, 18.5;
  38, 16.5;
  42, 16.5;
  42, 10;
  38, 10;
```

```
38.5, 10;
38.5, 8;
  39, 8;
  39, 4.5;
35.5, 0;
   2, 0;
   2, 8;
35.5, 8;
   0, 10;
 4.5, 10;
 4.5, 16.5;
   0, 16.5;
   4, 16.5;
 3.5, 22.5;
 6.5, 16;
 6.5, 10.5;
  36, 10.5;
  36, 16;
  22, 16;
26.5, 16;
  31, 16;
   4, 18.5;
 3.5, 18.5;
15.5, 0];

u1 = [...
40.5, 26.5
   7, 26.5
   7, 18.5
40.5, 18.5
  42, 16.5
  42, 10
  38, 10
  38, 16.5
38.5, 8
  39, 8
  39, 4.50
35.5, 4.50
   2, 0
   2, 10
35.5, 8
35.5, 0
 4.5, 10
 4.5, 16.5
   0, 16.5
   0, 10
```

```
   4, 18.5
   7, 22.5
 6.5, 10.5
  36, 10.5
  36, 16
 6.5, 16
  22, 10.5
26.5, 10.5
  31, 10.5
 3.5, 18.5
 3.5, 22.5
15.5, 8];

save('wallSet_santFloor.mat','u0','u1');
```

script_rayHost_multi.m

```
% script - rayHost_multi
%   performs raytracing simulation to produce sets of paths
   through which
%   propogation occurs between a set of transmitter locations
   (sx) and a
%   grid of receiver locations (specified by rgrid)
%
% requires:
%   wallSet.mat file, containing u0, and u1 variables (
   produced by
%   script_genSantFloor.mat)
%
% outputs:
%   K pathRecs files, where K is the number of transmitter
   locations
%

clearvars; % close all;

%% set up ray tracing parameters
N_rays = 2^11; % number of initial rays
N_trans = 8; % max number of ray interactions
r_seed = 0; % rng seed (used for minor random grid
   perturbation)
base_tag = 'sant_mglass'; % save file name
save_dir = 'prop_data/'; % data save directory
```

```
rgrid.r0 =[0 0]; % bottom left of receiver grid
rgrid.r1=[42 26.5]; % top right of receiver grid
rgrid.divisions_per_m_0 = 4; % receivers per meter in x
   direction
rgrid.divisions_per_m_1 = 4; % receivers per meter in y
   direction

% list of transmitter locations
% sx=[8.19 1.16;
%     2.46 18.21;
%     14.9 6.71;
%     34.1 15.3;
%     15.4 15.9;
%     18.1 4.1];
sx=[8.62 3.62;
    25.62 3.37;
    10.12 9.37;
    2.12 13.38;
    14.38 13.38;
    21.62 9.37;
    5.12 20.88;
    4.12 1.12];

rx_radius = 0.24; % simulated reciever radius (meters)
rx_max = 1000; % maximum paths per receiver
p_loss = 1; % amplitude path loss rate (1 in free space)
rx_pos_noise = 0.1; % stdev of position error (as a fraction
   of grid spacing)

load 'wallSet_santFloor.mat'; % load wall data

% load in p and s polarization wall profiles
[rws, rwp, tws, twp, theta_in] = fresnel_slab(1.0, 6.9);
% [rws, rwp, tws, twp, theta_in] = fresnel_slab(1.0, 60.0);
rw = 0.8*rws.';
tw = tws.';

%% display wall profile
figure; plot(1:size(rw),rw,1:size(tw),tw);
title('reflection/transmission wall profile');
legend({'reflection','transmission'});
xlabel('angle in degrees');
ylabel('coefficient value');
grid on;

%% initialize parameters
```

```
rng(r_seed); % set random seed
K = size(sx,1);
tag = ['s' num2str(K) '_' base_tag '_' num2str(rgrid.
   divisions_per_m_0) 'im_' ...
        num2str(log2(N_rays)) 'N' num2str(N_trans)];

walls = WallSet(u0,u1);

figure; plot(sx(:,1), sx(:,2), 'k+')
hold on; walls.plot();

%% set up receiver grid
rgrid.dim0 = floor((rgrid.r1(1)-rgrid.r0(1))*rgrid.
   divisions_per_m_0) - 1; % rgrid.dim0=3*2^6; % number of
   receivers per horiz. line
rgrid.dim1 = floor((rgrid.r1(2)-rgrid.r0(2))*rgrid.
   divisions_per_m_1) - 1; % rgrid.dim1=2^7; % number of
   recievers per vert. line
rgrid.dr = 1./[rgrid.divisions_per_m_0 rgrid.divisions_per_m_1
   ]; % grid spacing

rgrid.rg0 = ((0:rgrid.dim0-1)+0.5)*rgrid.dr(1); rgrid.rg1 =
   ((0:rgrid.dim1-1)+0.5)*rgrid.dr(2);
rgrid.rg0_full=repmat(rgrid.rg0,[rgrid.dim1 1]); rgrid.
   rg0_full=rgrid.rg0_full(:);
rgrid.rg1_full = repmat(rgrid.rg1.',[rgrid.dim0 1]);

dim = rgrid.dim0*rgrid.dim1;
rx = [rgrid.rg0_full rgrid.rg1_full];
rx = rx + rx_pos_noise*rgrid.dr.*randn(size(rx));

% Set up folder for saving
mkdir(save_dir,tag);

for k=1:K
    %% Perform ray propogation

    disp(['Performing Ray Propogation ' num2str(k) '/' num2str
       (K)]);
    tic;
    rs = RaySet(sx(k,:),N_rays,N_rays*(2^(N_trans+1)),N_trans)
       ;
    for t=1:N_trans+1
        rs=rs.propogate(walls, rw, tw);
    end
    toc;
```

```
    %% Get receive structure
    disp('Calculating receives...');
    tic;

    recs = rs.receive2D(rx,rx_radius,rx_max,p_loss);
    [del_spread, rms_del_spread] = recs2delSpd(recs);
    toc;

    figure; imagesc(rgrid.rg0,rgrid.rg1,reshape(recs.p,rgrid.
       dim1,rgrid.dim0)); title(['receive count map ' num2str(
       k)]);
    hold on; walls.plot(); colorbar;

%     figure; imagesc(rgrid.rg0,rgrid.rg1,reshape(del_spread,
   rgrid.dim1,rgrid.dim0)); title(['delay spread' num2str(k)])
   ;
%     hold on; walls.plot(); colorbar;

    figure; imagesc(rgrid.rg0,rgrid.rg1,reshape(rms_del_spread
       ,rgrid.dim1,rgrid.dim0)); title(['delay spread rms'
       num2str(k)]);
    hold on; walls.plot(); colorbar;

%     figure; imagesc(rgrid.rg0,rgrid.rg1,reshape(1./(
   rms_del_spread),rgrid.dim1,rgrid.dim0)); title(['coh bw'
   num2str(k)]);
%     hold on; walls.plot(); colorbar;

    save_name = [save_dir tag '/' 'pathRecs_' tag '_' num2str(
       k)  '.mat'];
    save(save_name,'recs','rx','sx','del_spread','rgrid','-v7
       .3');

end
```

script_multiRecs2H.m

```
% script multiRecs2H
%   converts sets of paths (made of delays and amplitudes) to
   transfer
%   functions (H) and autocorrelation functions (R). Produces
   two sets, one
%   for varying bandwidth/sampling-frequency (dfs) and the
   other for
```

```
%   varying center frequency (dfc).
%
% requires:
%   pathRecs file for each transmitter location to test (
   produced by
%   script_rayHost_multi.m)
%
% outputs:
%   M*K pathHR_dfs files, where M is the number of frequency
   configurations
%   for dfs, and K is the initial number of transmitters
%   M*K pathHR_dfc files, where M is the number of frequency
   configurations
%   for dfc, and K is the initial number of transmitters
%

clearvars; % close all;

load_dir = 'prop_data/'; % data load directory
save_dir = 'prop_data/'; % data save directory

% tag corresponding to desired dataset
tag = 's8_sant_mglass_4im_11N8';
% tag = 's8_sant_mglass_1im_9N4';

% Go to subfolder for tag
load_dir = [load_dir tag '/'];
save_dir = [save_dir tag '/'];

% Load one file to get configuration variables
load([load_dir 'pathRecs_' tag '_1' '.mat']);
S = size(sx,1);

%% convert raw receiver delays/gains into transfer functions/
   autocorrs

% Varying fs datatset
fc = 900e6; %400e6 900e6 2.4e9 5.0e9 %
fs = [5e5 1e6 2e6 4e6 8e6 16e6 32e6 64e6 128e6]; %
L_ac = fs*0+128;%max(16,ceil(fs*1e-6)); % length of ac
rel_bw=0.5;

M = length(fc);
N = length(fs);

[m, n] = meshgrid(1:M,1:N);
```

```
m = m(:).'; n = n(:).';
fc_full = repmat(fc(m), [1 S]);
fs_full = repmat(fs(n), [1 S]);
L_ac_full = repmat(L_ac(n), [1 S]);
MN = M*N;
s_full = sort(repmat(1:S, [1 MN]));

for s=1:S
    disp('Loading receive data');
    load([load_dir 'pathRecs_' tag '_' num2str(s) '.mat']);

    for m=1:MN

    disp(['Generating transfer functions and autocorrelations
       ... ' num2str(s) '/' num2str(S)]);
    disp(['Generating... ' num2str(m) '/' num2str(MN)]);

    t_start=tic;
    [H, R, R2] = recs2h(recs, L_ac_full(m), fc_full(m),
       fs_full(m), rel_bw);

    save([save_dir 'pathHR_' tag '_dfs_' num2str(m + (s-1)*MN)
        '.mat'],...
        'sx','rx', 'L_ac_full','fc_full','fs_full','s_full','
           L_ac',...,
        'fc','fs','H','R','R2','rgrid','-v7.3');

    toc(t_start);

    end
end

% varying fc dataset
fc = [100e6 400e6 900e6 1.2e9 2.4e9 5.0e9]; %
fs = [8e6 16e6 32e6]; %
L_ac = fs*0+128;%max(16,ceil(fs*1e-6)); % length of ac
rel_bw=0.5;
M = length(fc);
N = length(fs);

[m, n] = meshgrid(1:M,1:N);
m = m(:).'; n = n(:).';
fc_full = repmat(fc(m), [1 S]);
fs_full = repmat(fs(n), [1 S]);
L_ac_full = repmat(L_ac(n), [1 S]);
MN = M*N;
```

```
s_full = sort(repmat(1:S, [1 MN]));

for s=1:S
    disp('Loading receive data');
    load([load_dir 'pathRecs_' tag '_' num2str(s) '.mat']);

    for m=1:MN

    disp(['Generating transfer functions and autocorrelations
       ... ' num2str(s) '/' num2str(S)]);
    disp(['Generating... ' num2str(m) '/' num2str(MN)]);

    t_start=tic;
    [H, R, R2] = recs2h(recs, L_ac_full(m), fc_full(m),
       fs_full(m), rel_bw);

    save([save_dir 'pathHR_' tag '_dfc_' num2str(m + (s-1)*MN)
        '.mat'],...
        'sx','rx', 'L_ac_full','fc_full','fs_full','s_full','
           L_ac',...,
        'fc','fs','H','R','R2','rgrid','-v7.3');

    toc(t_start);

    end
end

disp('...finished');
```

script_eval_HR.m

```
% script - eval_HR
%   evaluates performance of fingerprinting algorithms with
   different
%   parameters and conditions, and produces summary plots.
%
% requires:
%   M*K pathHR_dfs files, where M is the number of frequency
   configurations
%   for dfs, and K is the initial number of transmitters (
   produced by
%   script_multiRecs2H)
%   M*K pathHR_dfc files, where M is the number of frequency
   configurations
```

```
%   for dfc, and K is the initial number of transmitters (
    produced by
%   script_multiRecs2H)
%
% outputs:
%   plots summarizing performance of channel fingerprinting
%

clearvars; % close all;

load_dir = 'prop_data/'; % data load directory
save_dir = 'prop_data/'; % data save directory
r_seed = 0; % rng seed (used for simulated estimation/noise
    effects)

% tag corresponding to desired dataset
tag = 's8_sant_mglass_4im_11N8';
% tag = 's8_sant_mglass_1im_9N4';



% Go to subfolder for tag
load_dir = [load_dir tag '/'];
save_dir = [save_dir tag '/'];

% Load first data for parameters
load([load_dir 'pathHR_' tag  '_dfs_1.mat']);
S = size(sx,1);
MN = length(fs_full)/S;

% apply seed
rng(r_seed);

% create default train/test grids
d1m=(0:4:rgrid.dim0-1);
d2m=(0:4:rgrid.dim1-1);
d1t=(1:2:rgrid.dim0-1);
d2t=(1:2:rgrid.dim1-1);
idx_m= 1+d1m*rgrid.dim1 + d2m.';
idx_t= 1+d1t*rgrid.dim1 + d2t.';
idx_m=idx_m(:); idx_t=idx_t(:);
if any(idx_m == idx_t.','all'), error('Testing on map data');
    end

% create large train/test grid
d1m_l=(0:2:rgrid.dim0-1);
```

```
d2m_l=(0:2:rgrid.dim1-1);
idx_ml= 1+d1m_l*rgrid.dim1 + d2m_l.';
idx_ml=idx_ml(:);
if any(idx_ml == idx_t.','all'), error('Testing on map data');
    end

% create small train/test grid
d1m_s=(0:8:rgrid.dim0-1);
d2m_s=(0:8:rgrid.dim1-1);
idx_ms= 1+d1m_s*rgrid.dim1 + d2m_s.';
idx_ms=idx_ms(:);
if any(idx_ms == idx_t.','all'), error('Testing on map data');
    end

% prepare variables with grids
rm = rx(idx_m,:);
rt = rx(idx_t,:);
rm_l = rx(idx_ml,:);
rm_s = rx(idx_ms,:);

dt2 = ((rt(:,1) - rm(:,1).').^2 + (rt(:,2) - rm(:,2).').^2).';
dlt2 = ((rt(:,1) - rm_l(:,1).').^2 + (rt(:,2) - rm_l(:,2).')
   .^2).';
dst2 = ((rt(:,1) - rm_s(:,1).').^2 + (rt(:,2) - rm_s(:,2).')
   .^2).';

load 'wallSet_santFloor.mat';
walls = WallSet(u0,u1);

% plot grid information
figure; subplot(311); plot(rx(:,1),rx(:,2),'g.',rm(:,1),rm
   (:,2),'.',rt(:,1),rt(:,2),'.');
hold on; walls.plot(); grid on;
legend({'unused grid position', 'reference measurement', '
   test/eval measurement'});
title('normal grid');
% xlim([min(rx(:,1)) max(rx(:,1))]); ylim([min(rx(:,2)) max(rx
   (:,2))]);
subplot(312); plot(rx(:,1),rx(:,2),'g.',rm_l(:,1),rm_l(:,2),'.
   ',rt(:,1),rt(:,2),'.');
hold on; walls.plot(); grid on;
title('double-size grid');
% xlim([min(rx(:,1)) max(rx(:,1))]); ylim([min(rx(:,2)) max(rx
   (:,2))]);
subplot(313); plot(rx(:,1),rx(:,2),'g.',rm_s(:,1),rm_s(:,2),'.
   ',rt(:,1),rt(:,2),'.');
```

```
hold on; walls.plot(); grid on;
title('half-size grid');
% xlim([min(rx(:,1)) max(rx(:,1))]); ylim([min(rx(:,2)) max(rx
   (:,2))]);

% display predicted grid spacing
grid_distm = max(median(diff(rm)));
grid_distt = max(median(diff(rt)));
grid_distm_l = max(median(diff(rm_l)));
grid_distm_s = max(median(diff(rm_s)));
disp([grid_distm; grid_distt; grid_distm_l; grid_distm_s])

% define some utility functions
split_complex = @(R) [real(R) imag(R)];
index_f2 = @(x, idx1, idx2) x(idx1,idx2);
apply_pulse_shape = @(r2,w,L) index_f2(conv(w, r2),1:size(r2
   ,1),floor((size(r2,2)+length(w))/2)+(1:L));

clear H R R2;

t_start = tic;
[proto_err, ~] = l2fp_eval(rm,rm,rt,rt,dt2);
err_l2(MN,S) = proto_err;
err_gp(MN,S) = proto_err;
err_m2(MN,S) = proto_err;
err_wn(MN,S) = proto_err;
H_err_gp(MN,S) = proto_err;
Hna_err_gp(MN,S) = proto_err;
fft_err_gp(MN,S) = proto_err;
na_err_gp(MN,S) = proto_err;
lg_err_gp(MN,S) = proto_err;
sg_err_gp(MN,S) = proto_err;
wg_err_gp(MN,S) = proto_err;
ws_err_gp(MN,S) = proto_err;
T=4;
nse_err_gp = cell(T,1);
nse_pmf_gp = cell(T,1);
for t=1:T
    nse_err_gp{t}(MN,S) = proto_err;
end


for s=1:S
    for m=1:MN
        disp(['Loading data... ' num2str(m+(s-1)*MN) '/'
           num2str(MN*S)]);
```

```
load([load_dir 'pathHR_' tag '_dfs_' num2str(m+(s-1)*
    MN) '.mat']);

Rx_na = R{1}./vecnorm(R{1},2,2);
Rx_na = Rx_na(:,1:min(size(Rx_na,2),128));
L = size(Rx_na,2);
Rx_na = (1:-1/L:1/L).*Rx_na;
Rx = abs(Rx_na);

Hx_na = H{1}./vecnorm(H{1},2,2);
Hx_na = Hx_na(:,1:min(size(Hx_na,2),128));
Hx = abs(Hx_na);

Rt = Rx(idx_t,:);
Rm = Rx(idx_m,:);

disp(['evaluating performance... ' num2str(m+(s-1)*MN)
     '/' num2str(MN*S)]);

%% base
tic;


[err_l2(m,s), pmf_l2] = l2fp_eval(Rm,rm,Rt,rm,dt2);
[err_gp(m,s), pmf_gp] = gpfp_eval(Rm,rm,Rt,rm,dt2);

% gp eval for different grid size (x3)
Rm_l = Rx(idx_ml,:);
Rm_s = Rx(idx_ms,:);
[lg_err_gp(m,s), lg_pmf_gp] = gpfp_eval(Rm_l,rm_l,Rt,
    rm,dt2);
[sg_err_gp(m,s), sg_pmf_gp] = gpfp_eval(Rm_s,rm_s,Rt,
    rm,dt2);

% gp eval for H
Ht = Hx(idx_t,:);
Hm = Hx(idx_m,:);
[H_err_gp(m,s), H_pmf_gp] = gpfp_eval(Hm,rm,Ht,rm,dt2)
    ;

% gp eval for fft R
Rm_fft = abs(fft(Rx_na(idx_m,:).')).';
Rt_fft = abs(fft(Rx_na(idx_t,:).')).';
[fft_err_gp(m,s) ,fft_pmf_gp] = gpfp_eval(Rm_fft,rm,
    Rt_fft,rm,dt2);
```

```
% gp eval for non-abs R
Rm_na = split_complex(Rx_na(idx_m,:));
Rt_na = split_complex(Rx_na(idx_t,:));
[na_err_gp(m,s) ,nabs_pmf_gp] = gpfp_eval(Rm_na,rm,
   Rt_na,rm,dt2);

% gp eval for non-abs H
Ht_na = split_complex(Hx_na(idx_t,:));
Hm_na = split_complex(Hx_na(idx_m,:));
[Hna_err_gp(m,s), H_pmf_gp] = gpfp_eval(Hm_na,rm,Ht_na
   ,rm,dt2);

% gp eval for noisy/low sample-rate (x4?)
H_mag = vecnorm(H{1},2,2);
H_mdn = 10*log10(median(H_mag));
N_t = [200000 200000 2000000 2000000];
snr_t = [20 40 20 40];
L = size(R{1},2);
for t=1:T
    R_nse = R{1};
    for k=1:size(Hx,1), R_nse(k,:) = (1:-1/L:1/L).*
       R_sim_noise(H{1}(k,:), snr_t(t)-H_mdn, N_t(t),
       L); end
    Rt_nse = R_nse(idx_t,:); Rt_nse=abs(Rt_nse)./
       vecnorm(Rt_nse,2,2);
    Rm_nse = R_nse(idx_m,:); Rm_nse=abs(Rm_nse)./
       vecnorm(Rm_nse,2,2);
    [nse_err_gp{t}(m,s), nse_pmf_gp{t}] = gpfp_eval(
       Rm_nse,rm,Rt_nse,rm,dt2);

end

% gp eval for mismatched AC (x4?)
Rt2 = R2{1}(idx_t,:);
x = -16:16;

rbw = 0.45;
w_gauss = exp(-0.5*(x.^2)/(rbw)^2);
w_srrc = srrc_ac(1e-8+x,1/rbw,0.25);
L = size(Rx,2);
Rt_wg = zeros(size(Rt));
for k=1:size(Rt2,1), Rt_wg(k,:) = apply_pulse_shape(
   Rt2(k,:),w_gauss,L); end
Rt_wg=abs(Rt_wg)./vecnorm(Rt_wg,2,2);
[wg_err_gp(m,s), wg_pmf_gp] = gpfp_eval(Rm,rm,Rt_wg,rm
   ,dt2);
```

```
        Rt_ws = zeros(size(Rt));
        for k=1:size(Rt2,1), Rt_ws(k,:) = apply_pulse_shape(
            Rt2(k,:),w_srrc,L); end
        Rt_ws=abs(Rt_ws)./vecnorm(Rt_ws,2,2);
        [ws_err_gp(m,s), ws_pmf_gp] = gpfp_eval(Rm,rm,Rt_ws,rm
            ,dt2);

        toc;

    end
end

toc(t_start);

%% Load and process data for varying fc

% Load first data for parameters
fs0 = fs;
fs_full0 = fs_full;
load([load_dir 'pathHR_' tag  '_dfc_1.mat']);
S = size(sx,1);
MN = length(fs_full)/S;

fc_err_gp(MN,S) = proto_err;
% loop over center frequencies
for s=1:S
    for m=1:MN
        disp(['Loading data... ' num2str(m+(s-1)*MN) '/'
            num2str(MN*S)]);
        load([load_dir 'pathHR_' tag '_dfc_' num2str(m+(s-1)*
            MN) '.mat']);

        Rx_na = R{1}./vecnorm(R{1},2,2);
        Rx = abs(Rx_na);

        Rt = Rx(idx_t,:);
        Rm = Rx(idx_m,:);

        [fc_err_gp(m,s), pmf_gp] = gpfp_eval(Rm,rm,Rt,rm,dt2);
    end
end

toc(t_start);

% store parameters for varied fc plot
dfc.fs = fs;
```

```
dfc.fc = fc;
dfc.fs_full = fs_full;
dfc.fc_full = fc_full;

fs_full = fs_full0;

%% Plots
p_base1 = sum(dt2<1^2,'all')/numel(dt2);
p_base5 = sum(dt2<5^2,'all')/numel(dt2);
err_base = eval_err([],dt2);
fsp = @(err, c) plot(log2(fs0*1e-6)-1, err, [c ' x']);
fsl = @(val, c) plot(log2(fs0*1e-6)-1, fs0*0+val, [c]);
fspm = @(err,c) plot(log2(fs0*1e-6)-1, median(err,2), [c '-'])
   ;
fcl = @(val, c) plot(log2(fc*1e-6)-1, fc*0+val, [c]);
fcp = @(err, c) plot(log2(fc*1e-6)-1, err, [c ' x']);
fcpm = @(err,c) plot(log2(fc*1e-6)-1, median(err,2), [c '-']);
f2m = @(structname, fld) field2mat(structname, fld);


%% final research plots
wallFig = figure; hold on;
txplt=plot(rm(:,1),rm(:,2),'+',rt(:,1),rt(:,2),'x');
% plot(rx(:,1),rx(:,2),'g.',rm(:,1),rm(:,2),'.',rt(:,1),rt
   (:,2),'.');
% plot(rx(:,1),rx(:,2),'g.',rm_l(:,1),rm_l(:,2),'.',rt(:,1),rt
   (:,2),'.');
% plot(rx(:,1),rx(:,2),'g.',rm_s(:,1),rm_s(:,2),'.',rt(:,1),rt
   (:,2),'.');
rxplt=plot(sx(:,1), sx(:,2), 'ko'); walls.plot();
legend([txplt; rxplt],{'reference location', 'test location',
   'receiver location'});
xlim([min(rx(:,1)) max(rx(:,1))]); ylim([min(rx(:,2)) max(rx
   (:,2))]);


figure; subplot(121);
pl0=fsl(err_base.p15,'k--'); hold on;
e=f2m(err_gp,'p15'); c='b'; fsp(e,c); pl1=fspm(e,c);
e=f2m(err_l2,'p15'); c='r'; fsp(e,c); pl2=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 1.5m error');
legend([pl0,pl1,pl2],{'baseline', 'gaussian process', 'L_2 1-
   nearest neighbor'});
subplot(122);
pl0=fsl(err_base.p25,'k--'); hold on;
```

```
e=f2m(err_gp,'p25'); c='b'; fsp(e,c); pl1=fspm(e,c);
e=f2m(err_l2,'p25'); c='r'; fsp(e,c); pl2=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 2.5m error');
% legend([pl0,pl1,pl2],{'baseline', 'gaussian process', 'L_2
   1-nearest neighbor'});

figure; subplot(121);
pl0=fsl(err_base.p15,'k--'); hold on;
e=f2m(err_gp,'p15'); c='b'; fsp(e,c); pl1=fspm(e,c);
e=f2m(H_err_gp,'p15'); c='r'; fsp(e,c); pl2=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 1.5m error');
legend([pl0,pl1,pl2],{'baseline','autocorrelation','impulse
   response'});
subplot(122);
pl0=fsl(err_base.p25,'k--'); hold on;
e=f2m(err_gp,'p25'); c='b'; fsp(e,c); pl1=fspm(e,c);
e=f2m(H_err_gp,'p25'); c='r'; fsp(e,c); pl2=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 2.5m error');
% legend([pl0,pl1,pl2],{'baseline','autocorrelation','impulse
   response'});

figure; subplot(121);
pl0=fsl(err_base.p15,'k--'); hold on;
e=f2m(sg_err_gp,'p15'); c='g'; fsp(e,c); pl1=fspm(e,c);
e=f2m(err_gp,'p15'); c='b'; fsp(e,c); pl2=fspm(e,c);
e=f2m(lg_err_gp,'p15'); c='r'; fsp(e,c); pl3=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 1.5m error');
legend([pl0,pl1,pl2,pl3],{'baseline', '2m ref measurement grid
   ',...
    '1m ref measurement grid','0.5m ref measurement grid'});
% legend([pl0,pl1,pl2,pl3],{'baseline', '2m reference
   measurement grid',...
%     '1m reference measurement grid','0.5m reference
   measurement grid'});
subplot(122);
pl0=fsl(err_base.p25,'k--'); hold on;
e=f2m(sg_err_gp,'p25'); c='g'; fsp(e,c); pl1=fspm(e,c);
e=f2m(err_gp,'p25'); c='b'; fsp(e,c); pl2=fspm(e,c);
e=f2m(lg_err_gp,'p25'); c='r'; fsp(e,c); pl3=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 2.5m error');
```

```
% legend([pl0,pl1,pl2,pl3],{'baseline', '2m ref measurement
   grid',...
%     '1m ref measurement grid','0.5m ref measurement grid'});
% legend([pl0,pl1,pl2,pl3],{'baseline', '2m reference
   measurement grid',...
%     '1m reference measurement grid','0.5m reference
   measurement grid'});

figure; c_t = 'brgk'; subplot(121);
pl0=fsl(err_base.p15,'k--'); hold on;
t=1; e=f2m(nse_err_gp{t},'p15'); c=c_t(t); fsp(e,c); pl1=fspm(
   e,c);
t=2; e=f2m(nse_err_gp{t},'p15'); c=c_t(t); fsp(e,c); pl2=fspm(
   e,c);
t=3; e=f2m(nse_err_gp{t},'p15'); c=c_t(t); fsp(e,c); pl3=fspm(
   e,c);
t=4; e=f2m(nse_err_gp{t},'p15'); c=c_t(t); fsp(e,c); pl4=fspm(
   e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 1.5m error');
legend([pl0,pl1,pl2,pl3,pl4],{'baseline', '200k meas, 20dB SNR
   ',...
    '200k meas 40dB SNR','2mil meas 20dB SNR','2mil meas 40dB
       SNR'});
subplot(122);
pl0=fsl(err_base.p25,'k--'); hold on;
t=1; e=f2m(nse_err_gp{t},'p25'); c=c_t(t); fsp(e,c); pl1=fspm(
   e,c);
t=2; e=f2m(nse_err_gp{t},'p25'); c=c_t(t); fsp(e,c); pl2=fspm(
   e,c);
t=3; e=f2m(nse_err_gp{t},'p25'); c=c_t(t); fsp(e,c); pl3=fspm(
   e,c);
t=4; e=f2m(nse_err_gp{t},'p25'); c=c_t(t); fsp(e,c); pl4=fspm(
   e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 2.5m error');
% legend([pl0,pl1,pl2,pl3,pl4],{'baseline', '200k meas, 20dB
   SNR',...
%     '200k meas 40dB SNR','2mil meas 20dB SNR','2mil meas 40
   dB SNR'});

figure; subplot(121);
pl0=fsl(err_base.p15,'k--'); hold on;
e=f2m(err_gp,'p15'); c='b'; fsp(e,c); pl1=fspm(e,c);
e=f2m(wg_err_gp,'p15'); c='r'; fsp(e,c); pl2=fspm(e,c);
e=f2m(ws_err_gp,'p15'); c='g'; fsp(e,c); pl3=fspm(e,c);
```

```
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 1.5m error');
legend([pl0,pl1,pl2,pl3],{'baseline', 'ideal white noise',...
    'simple gaussian pulse','RRC 25% excess BW'});
%     'simple gaussian pulse','root-raised-cosine 25% excess
   BW'});
subplot(122);
pl0=fsl(err_base.p25,'k--'); hold on;
e=f2m(err_gp,'p25'); c='b'; fsp(e,c); pl1=fspm(e,c);
e=f2m(wg_err_gp,'p25'); c='r'; fsp(e,c); pl2=fspm(e,c);
e=f2m(ws_err_gp,'p25'); c='g'; fsp(e,c); pl3=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 2.5m error');
% legend([pl0,pl1,pl2,pl3],{'baseline', 'ideal white noise
   ',...
%     'simple gaussian pulse','RRC 25% excess BW'});

figure; subplot(121);
pl0=fsl(err_base.p15,'k--'); hold on;
e=f2m(err_gp,'p15'); c='b'; fsp(e,c); pl1=fspm(e,c);
% e=f2m(H_err_gp,'p15'); c='r'; fsp(e,c); pl2=fspm(e,c);
e=f2m(na_err_gp,'p15'); c='r'; fsp(e,c); pl3=fspm(e,c);
% e=f2m(Hna_err_gp,'p15'); c='c'; fsp(e,c); pl4=fspm(e,c);
e=f2m(fft_err_gp,'p15'); c='g'; fsp(e,c); pl5=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 1.5m error');
legend([pl0,pl1,pl3,pl5],{'baseline', 'magnitude of elements',
   ...
    'augmented vector', 'spectral density'});
% legend([pl0,pl1,pl2,pl3,pl4],{'baseline', 'a',...
%     'b','c','d'});
subplot(122);
pl0=fsl(err_base.p25,'k--'); hold on;
e=f2m(err_gp,'p25'); c='b'; fsp(e,c); pl1=fspm(e,c);
% e=f2m(H_err_gp,'p25'); c='r'; fsp(e,c); pl2=fspm(e,c);
e=f2m(na_err_gp,'p25'); c='r'; fsp(e,c); pl3=fspm(e,c);
% e=f2m(Hna_err_gp,'p25'); c='c'; fsp(e,c); pl4=fspm(e,c);
e=f2m(fft_err_gp,'p25'); c='g'; fsp(e,c); pl5=fspm(e,c);
grid on; xlabel('bandwidth (2^x MHz)'); ylabel('proportion
   with less than 2.5m error');
% legend([pl0,pl1,pl3,pl5],{'baseline', 'magnitude of elements
   ',...
%     'augmented vector', 'spectral density'});

L_fs = length(dfc.fs);
L_fc = length(dfc.fc);
```

```
plc = cell(L_fs,1);
fc_err15_gp = f2m(fc_err_gp,'p15');
fc_err25_gp = f2m(fc_err_gp,'p25');
figure; subplot(121);
pl0=fcl(err_base.p15,'k--'); hold on;
for t=1:L_fs, e=reshape(fc_err15_gp(dfc.fs_full==dfc.fs(t)),
   L_fc,[]); c=c_t(t); fcp(e,c); plc{t}=fcpm(e,c); end
grid on; xlabel('center frequency (2^x MHz)'); ylabel('
   proportion with less than 1.5m error');
legend([pl0,plc{1},plc{2},plc{3}],{'baseline', '4 MHz
   bandwidth',...
    '8 MHz bandwidth','16 MHz bandwidth'});
subplot(122);
pl0=fcl(err_base.p25,'k--'); hold on;
for t=1:L_fs, e=reshape(fc_err25_gp(dfc.fs_full==dfc.fs(t)),
   L_fc,[]); c=c_t(t); fcp(e,c); plc{t}=fcpm(e,c); end
grid on; xlabel('center frequency (2^x MHz)'); ylabel('
   proportion with less than 2.5m error');
% legend([pl0,plc{1},plc{2},plc{3}],{'baseline', '4 MHz
   bandwidth',...
%     '8 MHz bandwidth','16 MHz bandwidth'});


disp('...finished');
save([save_dir 'eval_' tag '.mat'],'sx','rx', 'L_ac_full','
   fc_full','fs_full','L_ac','fc','fs','err_l2','err_gp','-v7
   .3');
```

B.3 Supporting functions

This section contains functions that are used by the previous scripts to construct channel impulse responses, perform fingerprinting steps, and evaluate performance.

delay2h.m

```
% function - delay2h
%   converts, set of path amplitudes and delays to transfer
   function, with
%   chosen sample frequency, center frequency, and bandwith
   relative to
%   sample frequency. Applies set number of samples to
   beginning and end of
%   minimum and maximum delay.
%
```

```
% inputs:
%   a: vector of path amplitudes
%   t: vector of path delays in seconds
%   fs: sample frequency
% optional inputs:
%   fc: center frequency of involved signals (affects phase)
%   rbw: relative bandwidth, channel response is band-limited
   to this
%        proportion. Recommended that 0.01 < rbw < 1.0.
%
% outputs:
%   h: channel impulse response as vector
%

function h=delay2h(a,t,fs,fc,rbw)
    if nargin<4, fc=0; end
    if nargin<5, rbw=1; end

    if isempty(t)
        h = zeros(10,1);
    else
        d=fs*t;
        h_L = max(round(d)) + 90; % 90 samples after end of
           last path
        n=(0:(h_L-1)) - 30; % 30 samples before first path
        h=sum(sinc((n-d(:))*rbw).*a(:).*exp(-2j*pi*fc*d(:)/3e8
           ),1).';

    end
end
```

eval_err.m

```
% function - eval_err
%   eval
%
% inputs:
%   pmf: probability mass function for probability of position
    (rows) from
%        data vs true position (columns)
%   dmt2: distance between test position (rows) and true
   position (columns)
%
% outputs:
```

```
%   err: struct containing error values for a variety of
   different metrics:
%     md: median estimated-position-to-true-position distance
%     d2: total estimated-position-to-true-position distance
   squared
%         normalized by total square distance between points
   in region
%     p05: proportion of estimated-position-to-true-position
   distances less
%          than 0.5 units
%     p10: as p05, but 1.0 units
%     p15: as p15, but 1.5 units
%     p20: as p20, but 2.0 units
%     p25: as p25, but 2.5 units
%     p30: as p30, but 3.0 units
%

function err = eval_err(pmf, dmt2)
if isempty(pmf)
    err=eval_err(zeros(size(dmt2))+(1/size(dmt2,1)),dmt2);
    err.md = sqrt(min(median(dmt2,2)));
else
    err.md = sum(sqrt(dmt2(pmf==max(pmf,2))))/size(dmt2,1);
    err.d2 = sum(dmt2.*pmf,'all')/sum(dmt2/size(dmt2,1),'all')
       ;
    err.p05 = sum(pmf(dmt2<0.5^2),'all')/size(dmt2,2);
    err.p10 = sum(pmf(dmt2<1.0^2),'all')/size(dmt2,2);
    err.p15 = sum(pmf(dmt2<1.5^2),'all')/size(dmt2,2);
    err.p20 = sum(pmf(dmt2<2.0^2),'all')/size(dmt2,2);
    err.p25 = sum(pmf(dmt2<2.5^2),'all')/size(dmt2,2);
    err.p30 = sum(pmf(dmt2<3.0^2),'all')/size(dmt2,2);
end
end
```

field2mat.m

```
% function - field2mat
%   extracts a particular field from a vector of structs to a
   vector
%
% inputs:
%   structArray: a vector of structs
%   field: name of field to extract form structArray
%
% outputs:
```

```
%   mat: vector of values extracted from given field of
   structArray
%

function mat = field2mat(structArray, field)
    mat = zeros(size(structArray));
    for k=1:numel(structArray), mat(k)=structArray(k).(field)
       (1);
end
```

fresnel_lossless.m

```
% function - fresnel_lossless
%   Generates reflection and transmission coefficients for an
   interface of
%   between two lossless dialectric materials.
%
% inputs:
%   mu: a tuple containing relative magnetic permeabilitiy
   (1.0 in many
%       materials) with the first value as the input media,
   and the second
%       as the output media
%   ep: a tuple containing relative electric permittivity (1.0
    for air,
%       higher number for many materials) with the first value
    as the input
%       media, and the second as the output media
%   theta_in: a vector of input angles (in radians) for which
   the
%             reflection and transmission coefficients should
   be calculated
%
% outputs:
%   rs: s or perpendicular polarized reflection coefficients
%   rp: p or parallel polarized reflection coefficients
%   ts: s or perpendicular polarized transmission coefficients
%   tp: p or parallel polarized transmission coefficients
%   theta_in: angle of incidence associated with each value in
    rs/rp/ts/tp
%   theta_tr: angle of incidence associated with leaving the
   interface
%   R: reflectance with fields s and p for the corresponding
   polarization
```

```
%   T: transmittance with fields s and p for the corresponding
    polarization
%

function [rs, rp, ts, tp, theta_in, theta_tr, R, T] =
   fresnel_lossless(mu, ep, theta_in)
if nargin<3, theta_in = linspace(0,pi/2, 90); end
if nargin==0, test_fresnel(); return; end
w = 1; % included in formula, but does not have effect when
   lossless
mu0 = 4e-7*pi;
ep0 = 8.854e-12;

% use relative permeability/permittivity
mu = mu*mu0;
ep = ep*ep0;

eta = sqrt(mu./ep);
beta = w*sqrt(mu.*ep);

% calculate transmitted angle
theta_tr = asin((beta(1)/beta(2))*sin(theta_in));

% fresnel equations (lossless)
rs = (eta(2)*cos(theta_in)-eta(1)*cos(theta_tr))...
     ./(eta(2)*cos(theta_in)+eta(1)*cos(theta_tr));

rp = (-eta(1)*cos(theta_in)+eta(2)*cos(theta_tr))...
     ./(eta(1)*cos(theta_in)+eta(2)*cos(theta_tr));

ts = (2*eta(2)*cos(theta_in))./(eta(2)*cos(theta_in)+eta(1)*
   cos(theta_tr));
tp = (2*eta(2)*cos(theta_in))./(eta(1)*cos(theta_in)+eta(2)*
   cos(theta_tr));

% correct for totat-internal-reflection
tir = abs(imag(theta_tr))>0;
ts(tir) = 0; tp(tir) = 0;
rs(tir) = 1; rp(tir) = -1;

% create reflectance and transmittance
t2T = (eta(1)*cos(theta_tr)./(eta(2).*cos(theta_in)));
R.s = real(rs).^2; R.p = real(rp).^2;
T.s = t2T.*real(ts).^2; T.p = t2T.*real(tp).^2;

end
```

fresnel_slab.m

```
% function - fresnel_slab
%   Generates reflection and transmission coefficients for a
   slab of
%   lossless dialectric material with up to several internal
   reflections,
%   and in air.
%
% inputs:
%   mu: relative magnetic permeabilitiy (1.0 in many materials
   )
%   ep: relative electric permittivity (1.0 for air, higher
   number for many
%       materials)
%
% outputs:
%   rs: s or perpendicular polarized reflection coefficients
%   rp: p or parallel polarized reflection coefficients
%   ts: s or perpendicular polarized transmission coefficients
%   tp: p or parallel polarized transmission coefficients
%   theta_in: angle of incidence associated with each value in
    rs/rp/ts/tp
%   theta_tr: angle of incidence associated with leaving the
   slab, should
%             match theta_in
%
%

function [rs, rp, ts, tp, theta_in, theta_tr] = fresnel_slab(
   mu, ep)
if nargin==0, test_fresnel_slab(); mu=1;ep=1; end
    [rs1, rp1, ts1, tp1, theta_in, theta_tr1] =
       fresnel_lossless([1 mu],[1 ep]);
    [rs2, rp2, ts2, tp2, theta_in2, theta_tr] =
       fresnel_lossless([mu 1],[ep 1], theta_tr1);

    rs = rs1 + ts1.*ts2.*(rs2);
    rp = rp1 + tp1.*tp2.*(rp2);

    ts = ts1.*ts2.*(1 + rs2.^2 + rs2.^4 + rs2.^6);
    tp = tp1.*tp2.*(1 + rp2.^2 + rp2.^4 + rs2.^6);
end
```

```
function test_fresnel_slab()
    % test with a glass slab
    mu = 1.0;
    ep = 1.5^2;
    [rs1, rp1, ts1, tp1, theta_in, theta_tr1] =
       fresnel_lossless([1 mu],[1 ep]);
    [rs2, rp2, ts2, tp2, theta_in2, theta_tr] =
       fresnel_lossless([mu 1],[ep 1], theta_tr1);
    [rs, rp, ts, tp, theta_in, theta_tr] = fresnel_slab(mu, ep
       );

    % plot results
    fresnel_plot(theta_in, rs1, rp1, ts1, tp1);
    fresnel_plot(theta_in2, rs2, rp2, ts2, tp2);
    fresnel_plot(theta_in, rs, rp, ts, tp);
end

function fresnel_plot(theta, rs, rp, ts, tp)
    figure; subplot(121); plot(theta, rs, theta, rp); grid on;
    xlabel('angle of incidence (radians)'); xlim([0,pi/2]);
    ylabel('reflection/transmission coefficient');
    subplot(122); plot(theta, ts, theta, tp); grid on;
    xlabel('angle of incidence (radians)'); xlim([0,pi/2]);
end
```

gp_mod2pmf_fast.m

```
% function - gp_mod2pmf_fast
%   Computes a normalized likelihood associated between a set
   of candidate
%   locations (rc) and a set of test features (Qt) using a set
    of reference
%   measurements/locations (Qm, qm) and Gaussian-process-model
    paramters
%   (mu, sig)
%
% inputs:
%   Qm: matrix of reference measurement feature vectors
%   rm: locations associated with measurement vectors
%   Qt: matrix of test measurement feature vectors for
   evaluation
%   rc: candidate locations where likelihood is evaluated (not
    necessarily
%       associated with Qt)
```

```
%   dmt2: pre-computed squared-distance between reference
   measurement and
%         test/candidate locations
%
% outputs:
%   pmf: probability mass function assigned by estimation
   process,
%        indicates estimated likelihood that a particular test
    measurement
%        (from Qt) came from a particular candidate location (
   from rc)
%   llh: log-likelihood, unnormalize log-likelihood used to
   generate pmf
%   m_cmap: mean of Gaussian posterior distributions used for
   estimation
%   sig_cmap: variance of Gaussian posterior distributions
   used for
%             estimation
%

function [pmf, llh, mu_cmap, sig_cmap] = gp_mod2pmf_fast(Qm,
   rm, Qt, rc, mu, sig)
% covariance proportion below this cutoff will be set to zero
   to save computation
limit_cutoff = 0.001;

  Qt = Qt - mu;
  Qm = Qm - mu;
  sig0 = max(sig(:,1),0.1*sig(:,2)); % set minimum sig0 (
     avoids degenerancy)
  sigk = sig(:,2);
  disk = sig(:,3);

  M = length(rm);
  T = length(rc);
  C = size(Qt, 1);
  L = size(Qm, 2);

  llh_tm = zeros(T, C);
  d2 = ((rm(:,1) - rm(:,1).').^2 + (rm(:,2) - rm(:,2).').^2);
  dt2 = ((rc(:,1) - rm(:,1).').^2 + (rc(:,2) - rm(:,2).').^2);

  mu_cmap = zeros(T,L);
  sig_cmap = zeros(T,L);
```

```
for l=1:L
  if abs(disk(l))>1e-16
      qml = Qm(:,l);
      qtl = Qt(:,l);

      % limit operations on distant points
      dist_cmp_limit = disk(l)*sqrt(-2*log(limit_cutoff));
      d_in_limit = (d2~=0 & d2<(dist_cmp_limit^2));

      % create sparse map covariance
      Sig_map = (sig0(l)+sigk(l))*eye(M);
      Sig_map(d_in_limit) = sigk(l)*exp((-0.5/(disk(l).^2))*
         d2(d_in_limit));

      % create test point vs map covariance
      dt_in_limit = (dt2~=0 & dt2<(dist_cmp_limit^2));
      if sum(dt_in_limit)==0, dt_in_limit = (dt2~=0 & dt2
         <(9*dist_cmp_limit^2)); end
      if sum(dt_in_limit)==0, continue; end
      Sig_tst2map = zeros(T, M);
      Sig_tst2map(dt2==0) = (0*sig0(l)+sigk(l));
      Sig_tst2map(dt_in_limit) = sigk(l)*exp((-0.5/(disk(l)
         ^2))*dt2(dt_in_limit));

      mu_cmap(:,l) = Sig_tst2map*cgs(Sig_map,qml
         ,[],[],[],[],qml);
      % mu_cmap(:,l) = Sig_tst2map*(Sig_map\qml); % non-cgs
         version

      pts_in_limit = sum(dt_in_limit,2);

      [max_p,max_t] = max(pts_in_limit);
      [min_p,min_t] = min(pts_in_limit);

      % calculate lowest variance and highest variance for
         gp posterior
      lim_t = dt_in_limit(max_t,:); %lim_t(max_t)=1;
      max_sig = sqrt(sig0(l)+sigk(l)-Sig_tst2map(max_t,lim_t
         )*cgs(Sig_map(lim_t,lim_t),Sig_tst2map(max_t,lim_t)
         .'));

      lim_t = dt_in_limit(min_t,:); %lim_t(min_t)=1;
      min_sig = sqrt(sig0(l)+sigk(l)-Sig_tst2map(min_t,lim_t
         )*cgs(Sig_map(lim_t,lim_t),Sig_tst2map(min_t,lim_t)
         .'));
```

```
        % fit variance to candidate points based on how many
           neigbhors
        % point has (this saves a lot of computation time and
           works well
        % when reference measurements are a grid)
        if max_p ~= min_p
            sig_cmap(:,l) = min_sig + (pts_in_limit-min_p)*((
               max_sig-min_sig)/(max_p-min_p));
        else
            sig_cmap(:,l) = max_sig; % special case
        end

        % calculate likelihood of all points using posterior
        llh_tm = llh_tm - 0.5*(log(2*pi*sig_cmap(:,l)) + ((qtl
           .' - mu_cmap(:,l))./sig_cmap(:,l)).^2);
    end
  end

  % move out of log domain and normalize to produce pmf
  llh = llh_tm;
  lh_tm = 1e-18+exp(llh_tm-max(llh_tm)); pmf=lh_tm./sum(lh_tm
     ,1);

end
```

gp_modEst1f.m

```
% function - modEst1f
%   calculates Gaussian-process-model parameters using a set
   of features
%   with associated positions
%
% inputs:
%   Q: matrix of features
%   p1: location coordinates associated with features
%       either contains both coordinates, or just x-coordinate
% optional inputs:
%   p2: location y-coordinate, optionally included as second
   vector
%
% outputs:
%   mu0: mean for Gaussian process model
%   s_opt: variance parameters for Gaussian process model,
%     first column: variance associated with individual
   measurement
```

```
%     second column: term associated with covariance size
   based on distance
%     third column: distance scale associated with covariance
%

function [mu0, s_opt] = gp_modEst1f(Q, p1, p2)
if nargin==2, p2=p1(:,2); p1=p1(:,1); end

K2 = 18; % Number of points used when estimating GP distance
   parameter

% lmvgpdf = @(x, mu ,Sig) -0.5*length(x)*log(2*pi) -0.5*logdet
   (Sig) - 0.5* ((x-mu).'*(Sig\(x-mu)));
d2 = (p1-p1.').^2 + (p2-p2.').^2;
L = size(Q, 2);
mu0 = mean(Q,1);
Q = Q - mu0;

% estimate typical distance between reference measurements
dp0 = (max(diff(sort(p1)))+max(diff(sort(p2))))/2;

% compute limits on allowed distance parameter, based on min/
   max covariance
min_var_frac = 0.01;
max_var_frac = 0.99;
max_cmp_dist = 5*dp0;

min_dk = sqrt(dp0/(-2*log(min_var_frac)));
max_dk = sqrt(dp0/(-2*log(max_var_frac)));

d2vs = d2(d2<max_cmp_dist^2);
knl_0 = double(d2vs==0);


s_opt = zeros(L,3);
for l=1:L
  ql = Q(:,l);
  cov_est = ql*ql.';
  cev = cov_est(d2<max_cmp_dist^2);
  cev0 = cov_est(d2==0);
  s0sk = mean(cev0);

  % create set of distances to check
  dkl = exp(linspace(log(min_dk),log(max_dk),K2));

  % evaluate GP models at different distance parameters
```

```
  s0 = zeros(K2,1);
  sk = zeros(K2,1);
  ek = zeros(K2,1);
  for kdk = 1:K2
    knl_dk = exp(-0.5*(d2vs/dkl(kdk).^2));
    sk(kdk) = max(0, (cev.'*knl_dk)/sum(knl_dk.^2));
    s0(kdk) = max(0, s0sk - sk(kdk));
    ek(kdk) = sum((s0(kdk)*knl_0 + sk(kdk)*knl_dk - cev).^2);
  end

  % identify smallest error param set
  [~, kmin] = min(ek);
  s0min = s0(kmin);
  skmin = sk(kmin);
  dklmin = dkl(kmin);

  if (1<kmin && kmin<K2)
    c2 = (ek(kmin-1)+ek(kmin+1))/2 - ek(kmin);
    c1 = (ek(kmin+1)-ek(kmin-1))/2;
%     c0 = ek(kmin);
    if c2>0
      ddk = dkl(2)-dkl(1);
      dklmin = dkl(kmin)-(ddk)*c1/(2*c2);
      knl_dk = exp(-0.5*(d2vs/dklmin.^2));

      skmin = max(0, (cev.'*knl_dk)/sum(knl_dk.^2));
      s0min = max(0, s0sk - skmin);
    end
  end
  s_opt(l,:)=[s0min skmin dklmin];

  % This code checks that the selected model performs better
     than a "null"
  % model, potentially improving performance, but has
     computational cost
  % M = length(d2);
  % Sig0 = (s0min*eye(M));
  % Sigk = skmin*exp((-0.5/(dklmin.^2))*d2);
  % Sig_fin = Sig0 + Sigk;
  % llh_alt = -0.5*logdet(Sig_fin)-0.5*(ql.'*(Sig_fin\ql));
  %
  % Sig_null = s0sk*eye(M);
  % llh_null = -0.5*logdet(Sig_null)-0.5*(ql.'*(Sig_null\ql));
  % if llh_null>llh_alt, s_opt(l,:) = [s0sk 0 0]; end

end
```

```
end
```

gpfp_eval.m

```
% function - gpfp_eval
%   evaluates performance of fingerprinting using a
%   principal-component-analysis feature, Gaussian process
   approach
%
% inputs:
%   Rm: matrix of reference measurement vectors
%   rm: locations associated with measurement vectors
%   Rt: matrix of test measurement vectors (for evaluation)
%   rt: candidate locations where likelihood is evaluated
%   dmt2: Pre-computed squared-distance between reference
   measurement and
%         test/candidate locations
%
% outputs:
%   err: struct containing error values for a variety of
   different metrics,
%        see eval_err for structure details
%   pmf: probability mass function assigned by estimation
   process,
%        indicates estimated likelihood that a particular test
    measurement
%        (from Rt) came from a particular test location (from
   rt)
%

function [err, pmf] = gpfp_eval(Rm, rm, Rt, rt, dmt2)
    [Qm, Qt] = reduce2q(Rm,Rt,1); % get PCA features
    [mu0, s_opt] = gp_modEst1f(Qm,rm); % estimate gp params
    [pmf,~]=gp_mod2pmf_fast(Qm,rm,Qt,rt,mu0,s_opt); % perform
       estimation

    pmf_m = 1e-64+(pmf==max(pmf,[],2));
    pmf_m = pmf_m./sum(pmf_m,1);

    err = eval_err(pmf_m,dmt2);

end
```

l2fp_eval.m

```
% function - l2fp_eval
%   evaluates performance of fingerprinting using basic l2-
   distance
%   approach
%
% inputs:
%   Rm: matrix of reference measurement vectors
%   rm(unused): placeholder for parity with gpfp
%   Rt: matrix of test measurement vectors (for evaluation)
%   rt(unused): placeholder for parity with gpfp
%   dmt2: pre-computed squared-distance between reference
   measurement and
%   test/candidate locations
%
% outputs:
%   err: struct containing error values for a variety of
   different metrics,
%        see eval_err for structure details
%   pmf: probability mass function assigned by estimation
   process,
%        indicates estimated likelihood that a particular test
    measurement
%        (from Rt) came from a particular test location (from
   rt)
%

function [err, pmf] = l2fp_eval(Rm, rm, Rt, rt, dmt2) % map

% K = length(rm);
[Km,~] = size(Rm);
[Kt,~] = size(Rt);

% ensure normalized vectors
Rm = gpuArray(Rm);
Rm = abs(Rm);
Rm = Rm./vecnorm(Rm,2,2);

Rt = gpuArray(Rt);
Rt = abs(Rt);
Rt = Rt./vecnorm(Rt,2,2);

% calculate l2 distances between reference measurements and
   test
% measurements
```

```
l2mt = gpuArray(zeros(Km,Kt));
for k1=1:Km
    dif = Rm(k1,:) - Rt;
    l2mt(k1,:) = sum(abs(dif).^2, 2);
end

% pmf assigns 1 to smallest distance position in rm
n0_pmf = 1e-64+(l2mt==min(l2mt,[],2));
n0_pmf = n0_pmf./sum(n0_pmf,1);

pmf = gather(n0_pmf);
err = eval_err(pmf,dmt2);


end
```

R_sim_noise.m

```
% function - R_sim_noise
%   produces an autocorrelation value simulating the effects
   of calculating
%   the autocorrelation from a limited number of samples (N)
   with additive
%   noise
%
% inputs:
%   h: true channel impulse response used to generate ac
   functions
%   snr: signal to noise ratio, specified in db,
%        so sig_pow/noise_pow = 10^(snr/10)
%   N: simulated number of samples used to estimate
   autocorrelation (higher
%      numbers will reduce variance)
%   L: desired length of autocorrelation
%
% outputs:
%   R: an autocorrelation vector simulating the effects of
   estimation from
%      a noisy signal with the given number of samples
%

function [R] = R_sim_noise(h, snr, N, L)
if nargin == 0, R = test_R_sim_noise();
else
    % set up power and scaling
```

```
    Lh = length(h);
    n_pow = 10^(-snr/10);
    l=(-L:L).';
    m_scal0 = N-abs(l);
    m_scal1 = 1./(N-abs(l)).^1;

    % create source signal term
    r_sig = sqrt(N^2)*(l==0) + sqrt(1+0.*(l==0)).*sqrt(m_scal0
       ).*(randn(2*L+1,1)+1j*randn(2*L+1,1));
    r_sig = 0.5*(r_sig + flip(conj(r_sig)));

    % create noise term
    r_nse = sqrt(N^2)*(l==0) + sqrt(1+0.*(l==0)).*sqrt(m_scal0
       ).*(randn(2*L+1,1)+1j*randn(2*L+1,1));
    r_nse = 0.5*(r_nse + flip(conj(r_nse)));

    % create cross term
    r_crs = sqrt(N)*conv(h,sqrt(0.5)*(randn(Lh+2*L+1,1)+1j*
       randn(Lh+2*L+1,1)));
    r_crs = r_crs(Lh:Lh+2*L);
    r_crs = sqrt(0.5)*(r_crs + conj(flip(r_crs)));

    % combine terms into autocorrelation
    r = conv(r_sig,conv(h(:),conj(flip(h(:)))));
    r = r(length(h):length(h)+2*L) + 2*sqrt(n_pow)*r_crs +
       n_pow*r_nse;
    r = (r.*m_scal1);

    % format result
    r=r.';
    R = r(1,L+2:2*L+1);
end
end

function R = test_R_sim_noise()
    K = 10000; % number of ac samples to generate
    L = 64; % length of ac sample
    rng(0); % repeatable seed (arbitrarily chosen)
    h = filter(hamming(8),1,randn(L*2+1,1)+1j*randn(L*2+1,1));
        % test impulse response
    snr = 10; % test snr

    r = zeros(K, L); ri = r;
    r_ref = zeros(K, L); ri_ref = r_ref;
```

```
    % generate samples using fast and standard generation
       methods
    tic;
    N = 4000;
    for k=1:K
        r(k,:) = real(R_sim_noise(h, snr , N, L));
        r_ref(k,:) = real(R_sim_noise_ref(h, snr , N, L));

        ri(k,:) = imag(R_sim_noise(h, snr , N, L));
        ri_ref(k,:) = imag(R_sim_noise_ref(h, snr , N, L));
    end
    toc;

    % plot summarizing images
    plotAcSummary(r); title('sim_r');
    plotAcSummary(r_ref); title('ref_r');
    plotAcSummary(ri); title('sim_i');
    plotAcSummary(ri_ref); title('ref_i');

R = [];
end

function [R] = R_sim_noise_ref(h, snr, N, L)
    R = (xcorr(filter(h,1,sqrt(0.5)*(randn(1,N)+1j*randn(1,N))
       ) + sqrt(10^(-snr/10))*sqrt(0.5)*(randn(1,N)+1j*randn
       (1,N)), L, 'unbiased'));
    R = R(1,L+2:2*L+1);
end

function f = plotAcSummary(r)
    Lo2 = size(r,2);
    f = figure; subplot(221); hold on; grid on;

    % plot quantiles with median (corresponding to integer std
       .
    % deviations for Gaussian random variable)
    plot(1:Lo2,quantile(r,0.0013));
    plot(1:Lo2,quantile(r,0.0228));
    plot(1:Lo2,quantile(r,0.1587));
    plot(1:Lo2,quantile(r,0.5));
    plot(1:Lo2,quantile(r,0.8413));
    plot(1:Lo2,quantile(r,0.9772));
    plot(1:Lo2,quantile(r,0.9987));

    er = r-median(r);
    subplot(222); hold on; grid on;
```

```
    % plot quantiles around median (corresponding to integer
       std.
    % deviations for Gaussian random variable)
    plot(1:Lo2,quantile(er,0.0013));
    plot(1:Lo2,quantile(er,0.0228));
    plot(1:Lo2,quantile(er,0.1587));
    plot(1:Lo2,quantile(er,0.5));
    plot(1:Lo2,quantile(er,0.8413));
    plot(1:Lo2,quantile(er,0.9772));
    plot(1:Lo2,quantile(er,0.9987));

    % plot image depicting covariance structure
    subplot(223); imagesc(cov(r));

    % plot samples of generated functions
    subplot(224); plot(1:Lo2,r(1:4,:));
end
```

recs2delSpd.m

```
% function - recs2delSpd
%   calculates delay spread associated with a set of paths
   according to two
%   standard formulations of delay spread
%
% inputs:
%   recs: a structure containing distance and gain values
   associated with
%         multipath channel components, produced by RaySet
   receive2D
%         function
%
% outputs:
%   delay_spread: computed delay spread associated with the
   channels
%                 provided
%   delay_spread_rms: computed rms delay spread associated
   with the
%                     channels provided
%

function [delay_spread, delay_spread_rms]=recs2delSpd(recs)

    % set minimum delay to zero, and setup variables
```

```
    recs.d = recs.d-min(recs.d);
    N = length(recs.p);
    c = 3e8;
    delay_spread = zeros(N,1);
    delay_spread2 = zeros(N,1);
    delay_spread_rms = zeros(N,1);

    % calculate delay spread and rms delay spread
    for n=1:N
        idx = ~isinf(recs.d(n,:));
        delay_spread(n) = sum((recs.d(n,idx)/c).*abs(recs.g(n,
           idx)))...
            /(1e-120+sum(abs(recs.g(n,idx))));
        delay_spread2(n) = sum((recs.d(n,idx)/c).^2.*abs(recs.
           g(n,idx)))...
            /(1e-120+sum(abs(recs.g(n,idx))));
        delay_spread_rms(n) = sqrt(delay_spread2(n)-
           delay_spread2(n)^2);
    end
end
```

recs2h.m

```
% function - recs2h
%   converts sets of paths (made of delays and amplitudes) to
   transfer
%   functions (H) and autocorrelation functions (R) at
   specified center
%   frequency (fc), sample frequency (fs), and relative
   bandwidth (rel_bw)
%
% inputs:
%   recs: a struct of paths associated with a set of receivers
    with:
%     g: matrix of complex gains associated with a particular
   receiver and
%        path
%     d: matrix of distances associated with a particular
   receiver and path
%   L_ac: length to which output ac functions are set
%   fc: center frequency for signals of interest, affects
   complex phase
%   fs: sample frequency for desired channel impulse responses
    and
%       autocorrelations
```

```
%   rel_bw: relative bandwidth of signal relative to Nyquist.
   Channel will
%          be bandlimited to specified value. (set to 1.0 for
    no
%          additional bandlimiting)
%
% outputs:
%   H: matrix of impulse responses associated with specified
   receives
%   R: matrix of one-sided autocorrelation functions from
   receives
%   R2: matrix of two-sided autocorrelation functions from
   receives
%

function [H, R, R2] = recs2h(recs, L_ac, fc, fs, rel_bw)
K=size(recs.d,1);

B = max([length(L_ac),length(fc),length(fs)]);
if(length(L_ac)==1), L_ac = repmat(L_ac,[B 1]); end
if(length(fc)==1), fc = repmat(fc,[B 1]); end
if(length(fs)==1), fs = repmat(fs,[B 1]); end

h = cell(K,1);
p = cell(K,1);

H = cell(B,1);
R = cell(B,1);
R2 = cell(B,1);

L_h=zeros(K,1);
for b=1:B

    disp(['Generating... ' num2str(b) '/' num2str(B)]);
    t_key = tic;
    for k=1:K
        [h{k}, p{k}]=recs2paths(recs, k, fc(b), fs(b), rel_bw)
           ;
        L_h(k)=length(h{k});
    end

    H{b} = zeros(K,max(L_h));
    R{b} = zeros(K,L_ac(b));
    R2{b} = zeros(K,2*L_ac(b)+1);

    for k=1:K
```

```
        % extend h to standardize length
        dL = max(L_h) - length(h{k});
        if dL>0
            h{k}=[h{k}; zeros(dL,1)];
        end
        H{b}(k,:) = h{k};
        R2{b}(k, :) = xcorr(h{k},L_ac(b));

    end

    R{b}(:, :) = R2{b}(:, L_ac(b)+2:end);
    toc(t_key);

end

end

function [h, path_set]=recs2paths(recs, idx, fc, fs, rel_bw)
    path_set.t = recs.d(idx,:)/3e8;
    path_set.a = recs.g(idx,:);
    p = recs.p(idx);

    % if no paths, apply dummy path
    if p==0
        path_set.a = 1e-12;
        path_set.t = 10;
        p=1;
    end

    % order based on arrival time
    [~,order] = sort(path_set.t);
    path_set.t=path_set.t(order);
    path_set.a=path_set.a(order);

    % keep only first p values (prunes NANs)
    path_set.t = path_set.t(1:p);
    path_set.a = path_set.a(1:p);

    path_set.t = (path_set.t-min(path_set.t)).';
    path_set.a = path_set.a.';

    % apply phase effect of path
    path_set.a = path_set.a.*exp(-2j*pi*fc*path_set.t);

    h = delay2h(path_set.a,path_set.t, fs, 0, rel_bw);
end
```

reduce2q.m

```
% function - reduce2q
%   converts a matrix of signal vectors into a matrix of
   feature vectors
%   obtained using principal-component-analysis.
%
% inputs:
%   aq: matrix containing a set of vectors to convert to PCA
   features
%   aqb: an additional matrix to convert to features, this
   matrix is not
%        used in computing feature transformation
%   t: a toggle for whether the input matrix should be
   transposed before
%      processsing, may be needed if input matrix isn't
   oriented as
%      expected
%
% outputs:
%   q: matrix that is featurized version of aq
%   qb: matrix that is featurized version of aqb
%   u: unitary matrix which extracts selected features when
   applied, also
%      contains vectors showing shape of features
%   mu: mean removed from aq before performing PCA
%

function [q, qb, u, mu]=reduce2q(aq, aqb, t)
  % ensure data is stacked right (tall vectors collected over
     columns)
  if nargin < 3
      if size(aq,2)<size(aq,1), t=1;
      else, t=0;
      end
  end
  if t==1, aq = aq.'; aqb = aqb.'; end

  % remove mean
  mu = mean(aq,2);
  aq = aq - mu;

  % identify principal components
  [U, S, ~] = svd(aq);
```

```
  p = cumsum(diag(S));
  M = min(find(p>0.99*p(end)));

  % extract components
  u = U(:,1:M).';
  q = u*aq;
  if nargin > 1, qb = u*(aqb-mu); % apply transform to aqb as
     well
  else,  qb = []; end
  if t==1, q = q.'; qb = qb.'; end
end
```

srrc_ac.m

```
% function - srrc_ac
%   produces vector containing samples of the autocorrelation
   associated
%   with a square-root-raised-cosine
%
% inputs:
%   t: vector of sample times at which srrc ac is sampled
%   w: width parameter, result is scaled widthwise by this
   value (default = 1)
%   alph: excess bandwidth associated with srrc, between zero
   and one
%
% outputs:
%   r: vector containing samples of srrc ac. Orientation will
   match input t
%
% test code:
%   figure; w=2; a=0.25; t=linspace(-3,3,1000); plot(t,(srrc(t
   ,w,a)))
%

function r = srrc_ac(t, w, alph)
    r= w*(sinc(t/w).*cos(alph*pi*t/w)./(1-(2*alph*t/w).^2) -
       alph/4 * sinc(alph*t/w).*cos(pi*t/w)./(1-(alph*t/w).^2)
       );
end
```

B.4 Ray-tracing classes

This section contains two classes which can be used to perform ray-tracing simulation for a two-dimensional environment. Their usage can be seen in the previous script "script_rayHost_multi.m".

WallSet.m

```
% class - WallSet
%   stores a set of wall segments describing a ray tracing
   environment.
%   Used in conjunction with raySet class
%
% further documentation provided throughout class
%

classdef WallSet
    properties
        % Convention: row = segment index, col = coordinate
           index (x,y)

        % Walls
        u0 = []; % contains first top-down 2D vertex of all
           wall segments
        u1 = []; % contains first top-down 2D vertex of all
           wall segments

        a = []; % top-down 2D normal vector for alternative
           representation
        b = []; % top-down 2D offset scalar for alternative
           representation

        wR = []; % wall reflectivity
        wT = []; % wall transmissivity

    end

    methods

        % constructor:
        %   creates a WallSet using a set of line segment
           start points (u0)
        %   and wall segment endpoints (u1)
        %
        function obj = WallSet(u0, u1)
```

```
        obj.u0=u0;
        obj.u1=u1;
    end

    % getters:
    %   L: gives wall count
    %   vertexList: gives back initial line segment
       endpoints
    %
    function L = W(obj) % wall count
        L = length(obj.u0);
    end

    function [u0, u1] = vertexList(obj)
        u0 = obj.u0;
        u1 = obj.u1;
    end

    % function - rayIntersect
    %   takes in a set of ray parameters (start point x0,
       direction v)
    %   and provides a list of collision information,
       where
    %   s is the ray displacement to the first
       intersection,
    %   w is the index of the wall first intersected, and
    %   a is the perpendicular vector to the wall first
       intersected.
    %
    function [s, w, a] = rayIntersect(obj, x0, v)
        x0 = gpuArray(x0);
        v = gpuArray(v);

        sR = zeros(length(x0),obj.W,'gpuArray');
        sW = zeros(length(x0),obj.W,'gpuArray');

        r = gpuArray(obj.u1-obj.u0);

        v_perp = flip(v,2);
        v_perp(:,1) = -v_perp(:,1);
        r_perp = flip(r,2);
        r_perp(:,1) = -r_perp(:,1);

        for w = 1:obj.W
            cu0=obj.u0(w,:);
            cu0=gpuArray(cu0);
```

```
                den = v_perp*r(w,:).';
                dif = x0 - cu0;
                % calculate distance along ray
                sR(:,w) = (dif*r_perp(w,:).')./den;
                % calculate position on wall
                sW(:,w) = (v_perp(:,1).*dif(:,1)+v_perp(:,2).*
                   dif(:,2))./den;
            end
            % remove intersections that do not fall on ray or
               wall segment
            sR(sW<0 | sW>1 | sR<0)=Inf;

            % Identify closest intersection
            [s, w] = min(sR.');

            % collect and return intersection information
            a = gather(r_perp(w,:)./(vecnorm(r_perp(w,:),2,2))
               );
            s = gather(s.');
            w = gather(w);

        end

        % function - plot
        %   plots wall object, normally with black lines,
           unless otherwise
        %   specified by optional format parameter
        %
        function plot(obj,fmt)
            if nargin<2, fmt='k'; end
            holdOn = ishold; hold on; % preserve hold status

            % stack up sets of walls
            plot_x=cat(2,obj.u0(:,1),obj.u1(:,1));
            plot_y=cat(2,obj.u0(:,2),obj.u1(:,2));

            % plot
            plot(plot_x.',plot_y.',fmt);

            if ~holdOn, hold off; end
        end
    end

end
```

RaySet.m

```
% class - RaySet
%   facilitates ray tracing simulation, stores rays and
   provides functions
%   for ray propogation
%
% further documentation provided throughout class
%

classdef RaySet
    properties
        DIM = 2; % Dimension count (2 or 3)
        N = 0; % max number of rays
        curN = 0; % current number of rays used
        T = 0; % max number of propogations
        t = 0; % current number of propogations
        x0 = []; % origin of ray at each time index
        v = []; % ray propogation vector at each time index
        gain = []; % path gain (will be less than 1)
        evts = []; % series of interaction events
        t0 = []; % time index where ray was generated
        tf = []; % time index when ray was terminated

    end

    methods

        % constructor:
        %   initializes RaySet with a source location (sx),
           initial number
        %   of rays (N0), max allowable number of rays (maxN),
            and maximum
        %   number of propogation steps (maxT)
        %
        function obj = RaySet(sx, N0, maxN, maxT)
            % initialize ray variables
            obj.N = maxN;
            obj.curN = N0;
            obj.T = maxT;
            obj.t = 1;
            obj.x0   = zeros(obj.N, obj.DIM, obj.T);
            obj.v    = zeros(obj.N, obj.DIM, obj.T);
            obj.gain = ones(obj.N, 1, obj.T);
            obj.evts = zeros(obj.N, 1, obj.T);
            obj.t0   = -1*ones(obj.N, 1);
```

```
        obj.tf   = obj.T*ones(obj.N, 1);

        % intialize radial rays from source point
        n=(1:N0)';
        obj.x0(n,:,1) = repmat(sx,[N0 1]);
        obj.t0(n,1) = ones(N0, 1);
        ang = ((0-2*pi)*(n-1)/N0);
        obj.v(n,:,1) = cat(2,cos(ang),sin(ang));


    end

    % function - propogate
    %   propogates current ray set through one additional
       wall
    %   intersection step, requires WallSet object (walls)
       , and
    %   reflection (R) and transmission (T) profile
       vectors. Note that
    %   if phase is flipped on reflection, this should be
       included as a
    %   signed value on R
    %
    function obj = propogate(obj, walls, R, T)
        if nargin<4, R=0.5*ones(91,1); T=0.5*ones(91,1);
           end

        activeIdx=(obj.t0>=0 & obj.tf>=obj.t);

        cx0=(obj.x0(activeIdx,:,obj.t));
        cv=(obj.v(activeIdx,:,obj.t));

        % s is displacement, w is wall index, a is normal
           vector
        [s, w, a] = ...
            walls.rayIntersect(cx0,cv);

        % update rays as reflections
        obj.x0(activeIdx,:,obj.t+1) = (cx0 + (s-1e-14).*cv
           );
        obj.evts(activeIdx,1,obj.t+1) = -w;

        th = (cv(:,1).*a(:,1)+cv(:,2).*a(:,2));
        th = max(ceil(acosd(abs(th))),1);
        rv = (cv - 2*(cv(:,1).*a(:,1)+cv(:,2).*a(:,2)).*a)
           ;
```

```
        obj.v(activeIdx,:,obj.t + 1) = rv./vecnorm(rv,2,2)
            ;

        obj.gain(activeIdx,:,obj.t + 1) = R(th).*obj.gain(
            activeIdx,:,obj.t);
        obj.tf(activeIdx) = obj.t + 1;

        % remove vectors leaving structure
        outRays = (isnan(s) | isinf(s));
        fndAct=find(activeIdx);
        outIdx = fndAct(outRays);
        obj.tf(outIdx) = obj.t;
        activeIdx(outIdx)=0;

        % create transmitted vectors
        newIdx=false(size(obj.t0));
        newIdx((obj.curN+1):obj.curN+sum(activeIdx))=1;

        obj.x0(newIdx,:,1:obj.t) = obj.x0(activeIdx,:,1:
            obj.t);
        obj.evts(newIdx,:,1:obj.t) = obj.evts(activeIdx
            ,:,1:obj.t);

        % update next vertex and event history
        th = th(~outRays);
        obj.x0(newIdx,:,obj.t+1) = (cx0(~outRays,:) + (s(~
            outRays)+1e-14).*cv(~outRays,:));
        obj.evts(newIdx,1,obj.t+1) = w(~outRays);
        obj.v(newIdx,:,obj.t + 1) = cv(~outRays,:);
        obj.t0(newIdx) = obj.t + 1;

        obj.gain(newIdx,:,obj.t + 1) = T(th).*obj.gain(
            activeIdx,:,obj.t);
        obj.tf(newIdx) = obj.t + 1;

        obj.curN=obj.curN+sum(newIdx);


        % update time index
        obj.t = obj.t + 1;

    end


    % function - receive2D
```

```
        %   compares contained rays to a set of reciever
            locations (rx) to
        %   check if they pass within a certain distance (
            radius). When
        %   this occurs, it is considered a "received path"
            and propogation
        %   distances and gains are stored in recs. A maxR
            value sets limit
        %   for receives per receiver (for memory management
            purposes) and
        %   gains are set using a specified path loss (
            pathLoss)
        %
        function [recs] = receive2D(obj, rx, radius, maxR,
            pathLoss)
            R = size(rx,1);
            recs.d=inf(R,maxR);
            recs.b=zeros(R,maxR);
            recs.g=(zeros(R,maxR));
            recs.p=(zeros(R,1)); % path count
%             rx=gpuArray(rx);

            for rt=1:obj.T-1
                activeIdx=sparse(obj.t0>=0 & obj.t0<=rt & obj.
                    tf>=rt+1);
                cx0=(obj.x0(activeIdx,:,rt));
                cxf=(obj.x0(activeIdx,:,rt + 1));

                vd=cxf-cx0;
                nvd=vecnorm(vd,2,2);
                uvd=vd./nvd;
                for cr = 1:R
                    cx0rx = rx(cr,:)-cx0; % get receiver
                        relative to ray source position
                    cx0nx = (uvd(:,1).*cx0rx(:,1)+uvd(:,2).*
                        cx0rx(:,2)).*uvd; % position of nearest
                         point relative to ray source
                    raySeg = cx0nx(:,1)./vd(:,1);
                    onRaySegment=(0<raySeg & raySeg<1);

                    nx = cx0nx +cx0;
                    dist=sum((nx-rx(cr,:)).^2,2);

                    inRadius=(dist<radius);
                    colIdx = (inRadius & onRaySegment);
```

```
                    fndAct=find(activeIdx);
                    colRays=fndAct(colIdx);

                    if ~isempty(colRays)
                        L = length(colRays);

                        cpx = (obj.x0(colRays,:,1:rt+ 1)); %
                           collision path coordinates
                        ceh = (obj.evts(colRays,:,1:rt)); %
                           collision event history
                        cg = (obj.gain(colRays,:,rt)); %
                           collision gain

                        chsum = sum(ceh(:,:,:),3);
                        cd = dist(colIdx); % collision
                           distance
                        nx = nx(colIdx,:);

                        % expunge paths with same history
                        for k=1:L
                            dropIdx=[];
                            chIdx=k+1:L;
                            chIdx=chIdx(chsum(k)==chsum(chIdx)
                               );
                            for l=chIdx% k+1:L

%                                 if (chsum(k)==chsum(l))
                                    if all(ceh(k,:,:)==ceh(l
                                       ,:,:))
                                        if cd(l)<cd(k) % if
                                           point l is closer,
                                           overwrite k
                                            cpx(k,:,:)=cpx(l
                                               ,:,:);
                                            ceh(k,:,:)=ceh(l
                                               ,:,:);
                                            nx(k,:,:)=nx(l
                                               ,:,:);
                                            cg(k,:,:)=cg(l
                                               ,:,:);
                                            cd(k)=cd(l);
                                        end
                                        dropIdx=[dropIdx l]; %
                                           #ok<AGROW>
                                    end
%                                 end
```

```
                    end
                    cpx(dropIdx,:,:)=[];
                    ceh(dropIdx,:,:)=[];
                    nx(dropIdx,:,:)=[];
                    cg(dropIdx,:,:)=[];
                    L=L-length(dropIdx);
                end

                if (L>maxR), error('Receive Limit
                   Exceeded'); end

                st=recs.p(cr)+1;
                recs.d(cr,st:L+st-1)=(sum(vecnorm(cpx
                   (:,:,2:end) - cpx(:,:,1:end-1),2,2)
                   ,3));
                recs.d(cr,st:L+st-1)=(recs.d(cr,st:L+
                   st-1) - (vecnorm(nx-cpx(:,:,end)
                   ,2,2)).');
                recs.g(cr,st:L+st-1)=sparse((recs.d(cr
                   ,st:L+st-1).^(-pathLoss)));
                recs.g(cr,st:L+st-1)=recs.g(cr,st:L+st
                   -1).*(cg(:,:,:).');
                recs.p(cr)=recs.p(cr)+L;

            end

        end
    end

    % reduce size to minimum necessary
    max_p = max(recs.p);
    recs.d=recs.d(:,1:max_p);
    recs.b=recs.b(:,1:max_p);
    recs.g=recs.g(:,1:max_p);
end


% function - plot
%   plots all rays along with a given WallSet,
   illustrating ray
%   operation. Avoid using this when too many rays or
   propogation
%   steps are used
%
function plot(obj, walls)
    holdOn = ishold; hold on; % preserve hold status
```

```
            for n=1:obj.curN
                plot_x = squeeze(obj.x0(n,1,obj.t0(n):obj.tf(n
                    )));
                plot_y = squeeze(obj.x0(n,2,obj.t0(n):obj.tf(n
                    )));
                plot(plot_x, plot_y);
            end
            if nargin>=2
                if ~isempty(walls), walls.plot(); end
            end

            if ~holdOn, hold off; end
        end
    end
end
```