An extensible toolkit for real-time high-performance wideband spectrum sensing

Bachelor Graduation Thesis

TU Delft — Electrical Engineering

Prof. G.J.T. Leus (Supervisor)
S.P. Chepuri (Daily Supervisor)
D.A. Dyonisius (Daily Supervisor)

W.P. Bruinsma
R.P. Hes
H.J.C. Kroep
T.C. Leliveld
W.M. Melching
T.A. aan de Wiel

July 4, 2015

Abstract

This document describes the design process of a software toolkit to perform high-performance wideband spectrum sensing. A prominent application of this is Cognitive Radio, a technique that aims to make more efficient use of the available radio spectrum. An extensive theoretical analysis will be performed. Various non-uniform sampling techniques will be discussed, such as coprime and circular sparse sampling. An algorithm to reconstruct the PSD of sub-Nyquist sampled signals will be developed and a detection algorithm which uses this PSD will be proposed. This analysis will be utilised to implement an extensible software toolkit written in Python. The software architecture and various design patterns that were utilised to structure the toolkit will be described and its quality and performance will be analysed. The hardware used for data acquisition, a USRP N210, will be introduced. The work will be concluded with a conclusion and its discussion.

Contents

1 | Preface

The past eight weeks have been interesting, to say the least. Being in the last year of the Bachelor phase of our studies, the only thing that separates us from our degree is the graduation project. For us this was a huge incentive to deliver, which we hope we did.

We have tried to combine an extensive mathematical analysis, software development and hardware design in this project, so that we could apply almost every aspect of our past education. The result is this thesis. Although it might seem a rather lengthy piece, we would like to encourage the reader to consider the extent of the covered subjects, as well as the amount of care with which it was put together.

It does not end there though. During the project we have also ventured out of our comfort zone and looked into the marketability of potential applications. We even touched upon the ethical implications of such applications. That subject is not included in this work, but will be presented separately as part of our business plan. It has, however, definitely influenced the path we set for ourselves during the course of the project.

All in all, we think the project was an instructive experience and hope that the following text reflects that, or even inspires the reader to continue the path we set out.

– The authors

2 | Introduction

Nowadays we cannot imagine a world without wireless technologies. The enormous increase in wireless applications has lead to a situation in which it has become increasingly difficult to find a piece of the wireless spectrum that is yet unused – or at least unlicensed – and therefore can be used for new technologies or to increase the capacity of existing ones. This so-called spectrum-scarcity problem makes it valuable to be able to gain insight in the current use of the radio spectrum in a fast and efficient way.

To alleviate the spectrum-scarcity problem the concept of Cognitive Radio (CR) has been introduced. A Cognitive Radio detects available channels in the spectrum that may be used for communication. The concept of a network of Cognitive Radios is that each Cognitive Radio gains access to the spectrum but does not interfere with other – licensed –users. Therefore it is important that large bands of the spectrum can be sensed such that it is feasible for each Cognitive Radio to find a free channel for communication. To reduce the interference with licensed users to a bare minimum, the Cognitive Radios should be able to not only sense large bandwidths, but also be able to sense those bandwidths as fast as possible. Sensing large bandwidths, however cannot be achieved with conventional techniques as those require Analog-to-Digital converters to operate at very high sampling rates. Analog-to-Digital converters running at such high sampling rate are power-hungry and cannot provide a feasible resolution in the band to be sensed. Therefore the need for other, revolutionary techniques arises.

In this thesis we present the implementation of a technology that can be used to accomplish just that by sampling at sub-Nyquist rates (i.e. requiring less samples than required by the Shannon-Nyquist sampling theorem). Obtaining samples at a lower sampling rate implies that cheaper and less energy-hungry hardware may be used to obtain the same results as with conventional sampling or, conversely, that using the same hardware, reconstruction of signals with higher bandwidths can be reconstructed than with conventional techniques. It goes without saying that these effects can be beneficial in a number of applications.

As required by our assignment, we have divided this work into two major parts, both of which are written by a separate set of authors and cover a different part of our total work. Before getting into either part, we will give a more in-depth description of the problem we are trying to solve, along with a general overview of our solution, which should help to put our work into the right context. This introduction should also enable the reader to understand both parts, independent of each other.

In part one, a theoretical introduction to the subject of wide band spectrum sensing is given, which should provide a mathematical basis in order to understand the inner workings of the used technologies. Algorithms will be formulated to reconstruct the power spectral density (PSD) of a signal that was sampled at a sub-Nyquist rate (reconstruction) and to distinguish any information that may be present in the reconstructed PSD from noise (detection). This theory is based on some very recent scientific works [1], [2].

Part two describes the implementation of these techniques using two Universal Software Radio Peripherals (USRPs), which are a type of software defined radio (SDR). The focus will be on the design of a software architecture that allows for high-performance digital signal processing using the USRPs and the design of a graphical user interface (GUI) that allows for visual verification and control of the entire system. The performance of the system will also be reviewed.

The thesis will be concluded by an overall conclusion and reflection.

3 | Project description

3.1 Problem statement

Cognitive Radio has been introduced to alleviate the problem of spectrum scarcity. At the heart of the Cognitive Radio lies spectrum sensing. Indeed, it is essential that Cognitive Radio's are able to sense a wide band of the spectrum to detect the presence of primary users. To minimise the interference with licensed users, it is crucial that this sensing process is as fast and as efficient as possible. Since Cognitive Radio networks are still in a conceptual stage, the need for a set of tools that can be used to sense a wide band with high efficiency arises.

These considerations lead us to our problem statement:

"Can we develop a toolkit that enables real-time high-performance wideband spectrum sensing?"

3.2 Proposed solution

The development of such a toolkit requires knowledge of algorithms that can be used to perform wideband spectrum sensing. Such algorithms do not only include compressive sensing, but also include detection algorithms that are used to detect the presence of users. Furthermore, to implement and effectively test those algorithms, it is necessary that they are implemented in a modular software solution. Furthermore, we require that our toolkit can be tested on a hardware platform. Based on the complexity of our solution, we decide to divide the design of our toolkit in two parts.

Part I of the thesis will provide a mathematical framework that solves the design problem at a theoretical level. The toolkit will be based upon this framework, whose implementation will be developed separately.

Part II of the thesis will discuss the design of a modular software architecture that implements the theoretical framework discussed in Part I. This software architecture will enable to test the solutions proposed by the theoretical framework on a hardware platform.

3.3 Source code and documentation

All the code and documentation we have written for this project can be found on GitHub at `https://github.com/pd0wm/spectral`.

4 | Similar research

Although the concept of a Cognitive Radio network is still at a conceptual stage, compressive sampling has been researched in for example [3]–[6]. Furthermore, recent developments have introduced techniques to perform compressive wideband sensing by noticing that in cognitive radio the signal content is not of interest, only its presence in the spectrum [2], [7].

The sampling techniques to sample signals at sub-Nyquist frequencies can be found in [1], [8], [9] and provide us with information on how to sample signals that can serve as the input for compressive spectrum sensing algorithms.

Detection of signals in general is a mature field, and lots of information can be found in textbooks concerning detection, for example [10], [11]. Detection for use in Cognitive Radio, is still an active area of research. A recent overview of techniques for spectrum sensing as a whole for use in Cognitive Radio is given by [12]. Other detection methods for use in cognitive radio can be found in [13], [14].

Although there is similar theoretical research available, we have only found a few details about practical implementations, which also motivates the distinction between the theoretical aspect and implementation in this thesis.

Part I

Theory

W.P. Bruinsma
H.J.C. Kroep
T.A. aan de Wiel

5 | Introduction

Noise is omnipresent. Even in the presence of signals, noise always will be there. Distincting between noise and signals forms an alluring problem; can good be distinguished from evil? One often asks where signals prevail in the ocean of noise. But this ocean is wide, as wide as the spectrum of every observed signal. And we do not have the power to sail this ocean. Therefore, we must be smart.

In the context of the spectrum of a signal, it is often asked which frequencies are occupied, and which are not. To solve this problem, one has to sense the spectrum. Spectrum sensing is determining which frequencies of a spectrum are occupied, and which are not. An occupied frequency means that it contains a signal distinct from noise. Therefore, the following problem arises. Given a signal, can we determine which frequencies of that signal are occupied with signals distinct from noise? In the introduction, we have seen that this question is asked in many applications.

This question however, is an answered question. By means of sampling and analysing the spectrum, one provides an answer. But this only works for frequencies relatively close together. It turns out that for frequencies far apart, spectrum sensing requires expensive hardware. And so a new question arises. Can we efficiently spectrum sense in the case of frequencies far apart? Wideband spectrum sensing tries to answer this very question. One then tries to determine which frequencies are occupied in a wide band.

Answering this question turns out to be hard. Consequently, wideband spectrum sensing is an active area of research, and there is no definitive answer yet.

In this part, we develop an approach to enable high-performance real-time wideband spectrum sensing. The part is divided into several chapters. We start with a detailed description of our approach, which we consider to be a system. Then, we give an overview of this system. In the overview, we look at the different system components, which will be described in separate chapters afterwards. Finally, we will evaluate the system as a whole.

6 System specification

6.1 General description

Our system is an approach which enables real-time high-performance wideband spectrum sensing. The goal of our system is to determine which frequencies of a spectrum contain signals distinct from noise. To achieve this goal, we will divide our system into several modules which all work together. In the following chapters each module will be designed separately.

Determining whether a signal is distinct from noise forms a decision problem. To evaluate this problem, one could reconstruct the spectrum of the signal, and determine whether it resembles one of noise. However, it can be shown that the decision problem is solved optimally[1] by evaluating the signal power [12]. Therefore, we could look per frequency at the signal power to decide whether a signal present or not. The signal power per frequency is also called the power spectral density. But it turns out that the power spectral density is derived from the spectrum of a signal, and requires less information than reconstruction of the spectrum would. Therefore, we focus on an approach which is centered around the reconstruction of the power spectral density.

The first step in high-performance real-time wideband spectrum sensing is sampling the signal of interest. Conventionally, this is done by sampling uniformly at the Nyquist frequency[2]. This ensures that the signal can be reconstructed after sampling. However, the Nyquist frequency becomes unreasonably high when one wants to sense a large bandwidth. As a consequence, expensive hardware must be used. Clearly, we must look for alternatives. To do this, we investigate sampling techniques which differ from uniform sampling.

So far, we have not specified what the system actually consists of. Concretely, we will design a mathematical framework. This framework will be made of many existing methods and theories, and will form the theoretical basis of the toolkit we are about to design. Since we design a mathematical framework, our specifications will be mostly functional, and less quantitative.

[1] This is only the case if no prior information is available.

[2] Remember that the Nyquist frequency is the minimum frequency at which a signal can be sampled without distorting the signal. The Nyquist frequency is twice the highest frequency present in the signal.

6.2 System specification

Now that we have identified our goal, we will proceed to describe our system in a detailed manner. This description consists of specifications according to which our system will be designed. The specifications are divided into several categories. The categories are then analysed by MoSCoW prioritisation[3]. We will be deliberately vague in quantizing our specifications, since most quantities are dependent on the application of the toolkit.

As stated in the introduction, the goal of our system is to determine which frequencies are occupied. However, some signals resemble noise very closely.[4] We therefore do not require that we recognise such signals. In addition, we do not require that we can sense an infinitely large bandwidth. In summary, our system *must* enable spectrum sensing such that the application

1. is able to identify at which frequencies signals are present, where signals
    a) can be any signal distinct from noise and
    b) can have any reasonable signal strength;
2. is able to sense any reasonably sized bandwidth and
3. the spectrum sensing is done in real-time.

In addition, our system *should* enable spectrum sensing such that the application

1. is able to sense very large bandwidths and
2. the spectrum sensing is done as efficient as possible.

Detection

We have described the functionality of our system on a high level. However, distinguishing signals from noise, which from now on we refer to as the detection of signals, requires a little more care. Ideally, we want to specify how accurate we can identify the frequencies which are occupied. Also, since wideband spectrum sensing is not considered solved, we want to know how accurate our detection is. In summary, detection of signals *must* be such that

1. detection of signals consists of determination of the set of frequencies which are occupied by signals other than noise;
2. the resolution of these frequencies can be specified and
3. the correctness of operation can be specified.

In addition, detection of signals *should* be such that

[3] MoSCoW prioritisation consist of dividing the specifications in *must-haves*, *should-haves*, *could-haves* and *would-haves*.

[4] These signals will be further discussed in Chapter 10.

1. the resolution of detection can be as high as possible;
2. its operation can be as correct as possible and
3. its operation is as efficient as possible.

Sampling techniques

We discussed that the first step in spectrum sensing is sampling the signal whose spectrum has to be sensed. It turned out that sampling at the Nyquist frequency is not efficient enough. We would therefore like to design efficient sampling techniques, which can distribute the workload across multiple devices. In summary the techniques used for sampling *must* allow for

1. correct operation of the detection;
2. usage of a single sampling device and
3. usage of multiple sampling devices such that the workload can be distributed across the devices.

In addition, the sampling techniques *should* allow for

1. sampling as efficient as possible.

Reconstruction

After the signal is sampled, the obtained information must be processed such that detection is possible. We already argued that for unknown signals, this is done most efficiently by reconstructing the power spectral density. Also, since we want to do spectrum sensing in real-time, this reconstruction must be done in real-time. Finally, the reconstruction must be as accurate as possible. In summary, the reconstruction *must* be able to

1. operate in conjunction with the sampling techniques and
2. operate in real-time.

In addition, the reconstruction *should* should be

1. as accurate as possible;
2. as efficient as possible and
3. as fast as possible.

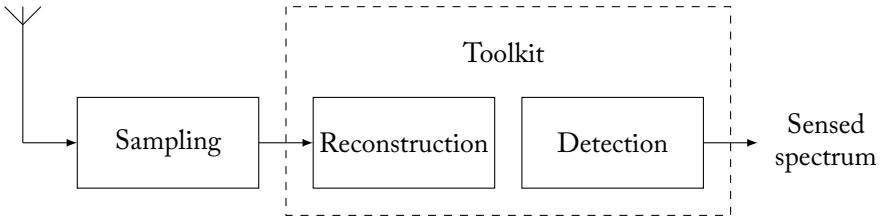


Figure 6.1: System overview. The system components are shown.

6.3 System overview

Our system consists of several components, which all work together to achieve the functionality specified in the previous section. The components are as follows.

Sampling Sampling consists of applying various sampling techniques to the signal whose spectrum has to be sensed. The actual sampling is not part of the system, but forms an interface between the signal and our system.

Reconstruction Reconstruction analyses the sampled signal and tries to reconstruct the power spectral density. Reconstruction has knowledge of the used sampling technique to perform its operation.

Detection Detection performs the identification of signals based on the result of the reconstruction. Detection and reconstruction closely work together.

Figure 6.1 shows a system overview. The dashed line depicts the system boundary.

In summary, the signal whose spectrum has to be sensed is sampled first. Then its power spectral density is reconstructed by means of algorithms designed for the sampling techniques. Detection then analyses this power spectral density, and decides per frequency whether a signal is present or not. This decision process is not based upon which kind of information is encoded in those frequencies, but only determines if *any* kind of information is present. After it is determined at which frequencies signals are present, the spectrum sensing process is complete.

Chapter 7 will give an introduction to non-uniform sampling. Then Chapter 8 will describe a way to reconstruct a power spectral density. Afterwards, different sampling techniques will be considered in Chapter 9. Chapter 10 will discuss how this power spectral density is used for detection. Finally, we will evaluate our system in Chapter 11 and discuss the obtained results in Chapter 12.

7 | Non-uniform sampling

7.1 Introduction

This chapter is an introduction to sampling concepts which will be key in understanding the design of our system.

7.2 Uniform sampling

As described by the system overview in Chapter 6, the first step in high-performance spectrum sensing consists of sampling the signal. Conventional methods sample at an uniform sampling rate, which means that the time between the instants the samples are taken is constant. Uniform sampling is illustrated in Figure 7.1. Here blocks represent samples such that subsequent rectangles are subsequent samples.

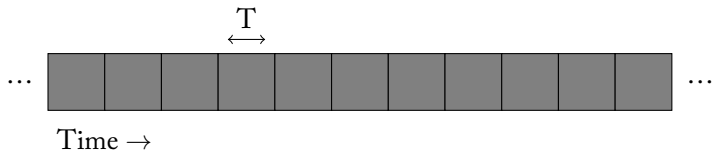


Figure 7.1: Uniform sampling with sampling period T

7.3 Non-uniform sampling

When one desires to ensure full reconstruction of an uniformly sampled signal, one needs to sample this signal at least at the Nyquist frequency. However, we are not necessarily interested in full recovery of the signal, since we only need to detect which frequencies of the signal are occupied by signals distinct from noise. In our sampling methods, we aim to not restrict ourselves to uniform sampling, and try to exploit non-uniform sampling. This allows for methods which sample the signal in a more efficient way. Since we aim to solely obtain the power spectral density of the signal, it turns out that by non-uniform sampling, we can still do this. Non-uniform sampling is illustrated in Figure 7.2. This figure is similar to Figure 7.1, but some blocks are left white. Only the grey blocks represent sample moments of the non-uniform sampler.

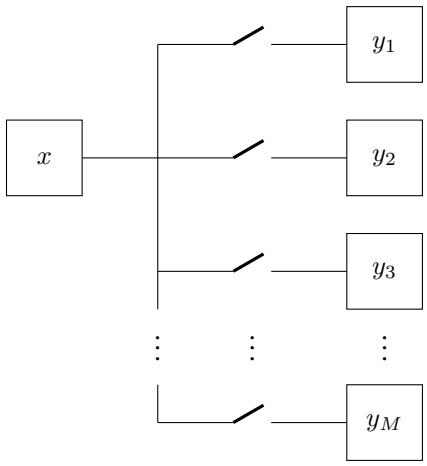


Figure 7.3: Conceptual schematic representation of multi-coset sampling with M cosets

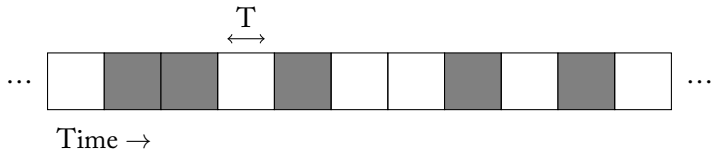


Figure 7.2: Non-uniform sampling with sampling period T. Grey blocks represent sample moments of the non-uniform sampler.

7.4 Multi-coset sampling

Non-uniform sampling, however, is not trivial to implement. To perform non-uniform sampling, we introduce a concept called multi-coset sampling. Multi-coset sampling is the name for a set of sampling methods that use multiple samplers which sample the same signal. These samplers are also called cosets. From now on, we will use these two terms interchangeably. Multi-coset sampling is illustrated in Figure 7.3. Here x represents the input signal, the switches represent a sample operation and y_i represent the outputs of the different cosets. In Chapter 8, we will deduce criteria that will further specify the samplers. Afterwards, in Chapter 9, we will reconsider non-uniform sampling in more detail.

8 | Reconstruction

8.1 Introduction

In the introduction, we argued that we want to avoid sampling at the Nyquist frequency. In the previous chapter, we introduced the concept of multi-coset sampling. Therefore, it would be nice if the cosets in multi-coset sampling could sample at sub-Nyquist frequencies. The samples obtained by these cosets could then be used to reconstruct the power spectral density of the sampled signal, which, once more, we argued to be important during detection.

This chapter introduces a method to reconstruct the power spectral density of a signal which is sampled at sub-Nyquist frequencies. Usually, sampling at sub-Nyquist frequencies causes aliasing, which means that the obtained signal is distorted. Our reconstruction method should be able to reconstruct the power spectral density of the signal in real-time, despite the fact that the sampled signal is aliased. We focus on an accessible, yet complete explanation of such a reconstruction method.

First of all, we will consider various methods which allow for reconstruction of the power spectral density of signals sampled at sub-Nyquist frequencies. We will then, based on considerations developed in the next paragraph, choose one of these methods. Next, we will discuss an algorithm based on this method. Afterwards, we will discuss some details which turn out to be essential when implementing the algorithm.

8.2 Overview

There are various methods to reconstruct the power spectral density of a signal. These methods can be roughly divided in two groups: methods based on signal reconstruction, such as [3], [4], [6], [8], [15]–[18], and methods which are not based on signal reconstruction, such as [1], [2].

Methods of the first group are based on the assumption that the signal is either sparse, or contains few frequencies[1]. Based on this assumption, they try to recover

[1]More specifically, methods of the first group are based on the assumption that the signal is sparse in a certain basis. Let $\mathbf{x}$ denote the signal sampled at the Nyquist-frequency. Also, let the matrix $\mathbf{\Phi}$ consist of rows of the identity matrix such that $\mathbf{y} = \mathbf{\Phi x}$ represents the signal sampled at a sub-Nyquist frequency. If $\mathbf{x}$ is sparse in the basis $\mathbf{\Psi}$, then there exists a sparse vector $\mathbf{s}$ such that $\mathbf{x} = \mathbf{\Psi s}$. Given $\mathbf{y}$, the original signal $\mathbf{x}$ can be obtained by finding the most sparse vector $\hat{\mathbf{s}}$ such that $\mathbf{y} = \mathbf{\Phi\Psi}\hat{\mathbf{s}}$. Then $\mathbf{x} = \mathbf{\Psi}\hat{\mathbf{s}}$. This problem can be formulated as an l_1-optimisation problem. Since it is assumed that the signal contains few frequencies, $\mathbf{\Psi}$ is often chosen to be the discrete Fourier transform matrix.

the signal sampled at sub-Nyquist frequencies. This can be done by optimisation[2] or by various algorithms[3]. There are a few problems with this approach. First, the assumption that the signal is either sparse or contains few frequencies may be not true in our case, since we did not restrict the signal which the application must be able to process. Second, solving an optimisation problem or making use of such an algorithm in real-time yields a computationally expensive and complex reconstruction method. In contrast, methods of the second group are not based on the assumption the signal is either sparse or contains few frequencies. They do not aim to reconstruct the signal, but try to solely reconstruct the power spectrum density. This allows for a computationally less expensive and less complex reconstruction method, but which is sufficient for detection purposes

Based on the stated reasoning, we decide to discuss a reconstruction method based on the method discussed in [2].

8.3 Reconstruction of the power spectral density

This section will discuss an algorithm to estimate in real-time the power spectral density of a signal which is sampled at sub-Nyquist frequencies. We consider multi-coset sampling such as described in Chapter 7. A step-by-step derivation of the algorithm in conjunction with more details on multi-coset sampling can be found in Appendix B.

Mathematical formulation of multi-coset sampling

Let the input signal sampled at the Nyquist frequency be denoted by $x[n]$. Let the number of cosets be given by M. Let the output of coset i be denoted by $y_i[n]$. Our reconstruction method estimates the power spectral density of $x[n]$ by making use of the outputs of all cosets. The Wiener-Khinchin theorem shows that estimating the power spectral density of $x[n]$ is equivalent to estimating the autocorrelation of $x[n]$.[4] Therefore, our reconstruction method aims to estimate the autocorrelation of $x[n]$. Let the autocorrelation of the input signal be denoted by $r_x[m]$. Let the cross-correlation of the output of cosets i and j be denoted by $r_{y_i,y_j}[m]$. We will make use of $r_{y_i,y_j}[m]$ to estimate $r_x[m]$. This estimation is based on the relationship

$$\mathbf{r}_y = \mathbf{R}\mathbf{r}_x.$$

Here $\mathbf{r}_y$ represents an aggregation of $r_{y_i,y_j}[m]$ and $\mathbf{r}_x$ represents an aggregation of r_x. We see that we can use $\mathbf{R}$ to relate $r_x[m]$ to $r_{y_i,y_j}[m]$. This means that $\mathbf{R}$ represents the relationship between the autocorrelation of the input signal and the cross-correlations of the outputs of the cosets. A visual representation of the relationship of all variables is given in Figure 8.1.

[2] If the signal is assumed to be sparse, then l_1-optimisation is often used. Such optimisation consists of minimising the l_1-norm of a vector, which is the sum of the absolute value of the elements of the vector.

[3] If the signal is assumed to contain few frequencies, then the MUSIC algorithm can be used [8].

[4] More specifically, the Wiener-Khinchin theorem states that the Fourier transform of the autocorrelation equals the power spectral density, which yields the equivalence.

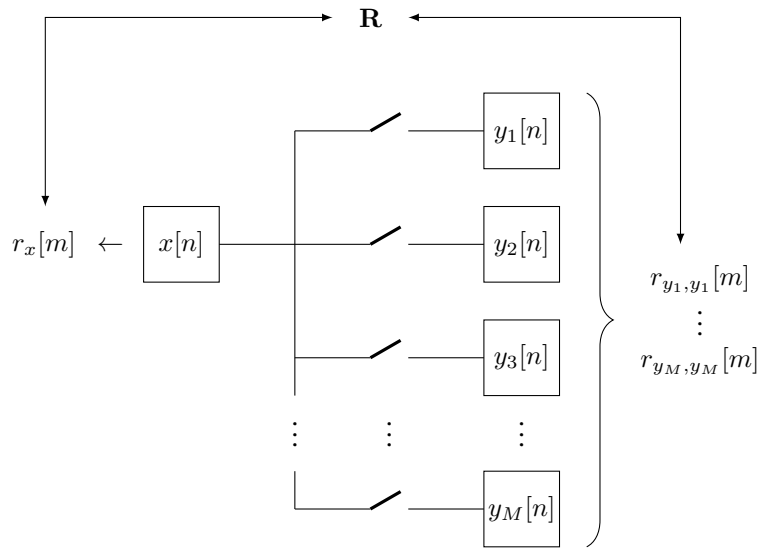


Figure 8.1: Visual representation of the relationship between the autocorrelation of the input signal and the cross-correlations of the output signals of the cosets

Sampling signals

Before we describe the algorithm, we have to study the output of a coset more closely. Note that the following paragraph will be a little more technical.

Every coset i is associated with a sampling signal $c_i[n]$. Also, every coset samples the input signal such that it is an N-decimation of the input signal. However, there may be an offset in time between the decimations of the different cosets. This concept is illustrated in Figure 8.2. If we divide the input signal in groups of N samples, then the output of a coset consists of the same sample of every group of N samples. The sampling signal of coset i is such that $c_i[n] = 1$ if coset i samples the $N - n$'th sample of every group of N samples of the input signal. This concept is discussed carefully in Appendix B. Also, this concept will be discussed in Chapter 9. From now on, we will refer to N as the downsampling factor.

Algorithm

It is clear that $y_i[n]$ is part of the input of the algorithm. The way all cosets sample the input signal, however, is determined by the sampling technique. The configuration refers to the way all cosets sample the input signal. Possible configurations are discussed in Chapter 7 and will be further discussed in Chapter 9.

The input of the algorithm also consists of the parameters M, N, L and K. Here M and N have already been discussed, where M represents the number of cosets and N the downsampling factor. The parameter L limits $r_{y_i,y_j}[m]$ in support from $m = -L$ to $m = L$.[5] This influences the length of the estimated autocorrelation,

[5] If $r_{y_i,y_j}[m]$ is limited in support from $m = -L$ to $m = L$, then this means that $r_{y_i,y_j}[m] = 0$ for $|m| > L$.

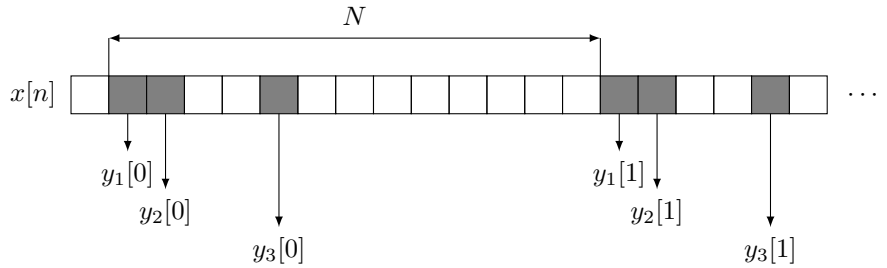


Figure 8.2: Illustration of how $y_i[n]$ is constructed from $x[n]$. The blocks represent the samples of $x[n]$.

Type	Parameter	Description
Input	M	Number of cosets
Input	N	Downsampling factor. Every coset is an N-decimation the input signal. However, there may be offsets between the decimations of different $y_i[n]$.
Input	$y_i[n]$	Output of coset i
Input	L	Support of $r_{y_i,y_j}[m]$. This parameter influences the resolution of the estimated power spectral density of $x[n]$.
Input	K	Accuracy factor. This parameter influences the accuracy of the estimated power spectral density of $x[n]$, but increases the measurement time. The measurement time consists of the time required to obtain KL samples of the output of every coset.
Output	$r_x[m]$	Autocorrelation of $x[n]$. The autocorrelation $r_x[m]$ is estimated for $\|m\| \leq LN$.

Table 8.1: Input and outputs of the reconstruction algorithm

which then determines the resolution of the estimated power spectral density of $x[n]$. The autocorrelation $r_x[m]$ will be estimated for $|m| \leq LN$. The parameter K influences the measurement time, which determines the accuracy of the estimated power spectral density. The measurement time consists of the time required to obtain KL samples of the output of every coset.

The inputs and output of the algorithm are summarised in Table 8.1. The algorithm consists of following steps.

Step 1: Determine $c_i[n]$ for every coset i.
The configuration determines $c_i[n]$ for every coset i. How $c_i[n]$ can be obtained from the configuration is explained in Chapter 9. Sufficient requirements on $c_i[n]$ are discussed in Section 8.4.

Step 2: Construct $\mathbf{R}$.
Section 8.4 explains how to construct $\mathbf{R}$.

Step 3: Measure $y_i[n]$ for KL samples for every coset i.

Step 4: Estimate $r_{y_i,y_j}[m]$ for every combination of cosets i and j.
Section 8.4 discusses how KL samples of $y_i[n]$ can be used to estimate $r_{y_i,y_j}[m]$.

Step 5: Construct $\mathbf{r}_y$.
If

$$\mathbf{r}_{y_i,y_j} = \begin{bmatrix} r_{y_i,y_j}[L] & \cdots & r_{y_i,y_j}[-L] \end{bmatrix}^T,$$

then

$$\mathbf{r}_y = \begin{bmatrix} \mathbf{r}_{y_1,y_1} \\ \vdots \\ \mathbf{r}_{y_M,y_M} \end{bmatrix}.$$

Step 6: Calculate $\mathbf{r}_x = \mathbf{R}^\dagger \mathbf{r}_y$, which yields an estimation of $r_x[m]$.
Here

$$\mathbf{r}_x = \begin{bmatrix} r_x[LN] & \cdots & r_x[-LN] \end{bmatrix}^T.$$

This means that the autocorrelation $r_x[m]$ is estimated for $|m| \leq LN$, as stated earlier. The matrix $\mathbf{R}^\dagger$ denotes the Moore-Penrose pseudoinverse[6] of $\mathbf{R}$.

8.4 Implementational details

This section will discuss some details of the algorithm which should be considered when implementing the algorithm. We assume that multi-coset sampling is used.

Requirements of the sampling signals

This subsection discusses sufficient requirements on the sampling signal $c_i[n]$ which yield a correct estimation. Remember that the sampling signal determines the way a coset samples the input signal. The derivation of these restrictions can be found in Appendix B. The requirements are as follows:

1. Every $c_i[n] = 0$ for $n < 0$ and $n > N - 1$.
2. All $c_i[n] = 1$ for a single $n = n_i$ and otherwise zero.
3. The circular sparse ruler problem, which is stated below, must be satisfied.

Circular sparse ruler problem Consider n_i for every coset i as discussed in the second requirement. The set consisting of $|n_i - n_j|$ and $N - |n_i - n_j|$ for all combinations of cosets i and j must make up $0, \ldots, N - 1$.

The circular sparse ruler problem will be further discussed in Chapter 9.

[6]The Moore-Penrose pseudoinverse is a generalisation of the inverse of a matrix and can be used to solve linear systems in the least-squares sense.

Estimation of the cross-correlations

In Step 4 of the algorithm we have to estimate $r_{y_i,y_j}[m]$. Suppose that $y_i[n]$ is known for $n = 0, \ldots, KL-1$. We assumed that $r_{y_i,y_j}[m] = 0$ for $|m| > L$. It then remains to estimate $r_{y_i,y_j}[m]$ for $|m| \leq L$. An estimator of $r_{y_i,y_j}[m]$ is given by

$$\hat{r}_{y_i,y_j}[m] = \frac{1}{KL - |m|} \sum_{k=l}^{u} y_i^*[k] y_j[k+m] \tag{8.1}$$

where $l = -\max\{0, m\}$ and $u = KL-1-\min\{0, m\}$ [19]. Notice that $E(\hat{r}_{y_i,y_j}[m]) = r_{y_i,y_j}[m]$, which means that $\hat{r}_{y_i,y_j}[m]$ is an unbiased estimator of $r_{y_i,y_j}[m]$.

Sparsity

Operations on large matrices can be computationally expensive. If a matrix is sparse[7], then functions designed for sparse matrices can be used. These functions can provide significant speedups. In the discussion on the unicity of the estimation in Appendix B, we argued that every row of $\mathbf{R}$ has exactly one nonzero element. Remember that $\mathbf{R}$ describes the relationship between the autocorrelation of the input signal and the observed quantities. Since $\mathbf{R}$ has $2LN+1$ columns, the fraction of nonzero elements of $\mathbf{R}$ is given by

$$\rho_{\text{nz}} = \frac{1}{2LN+1}.$$

Since LN determines the resolution of the estimated power spectral density of the input signal, the product LN is usually chosen large. This means that $\mathbf{R}$ is usually a sparse matrix. This property will be exploited Section 21.2.

Efficient generation of the matrices

In Step 2 of the algorithm, $\mathbf{R}$ has to be constructed. It turns out that $\mathbf{R}$ can be generated efficiently using commonly available functions. To generate $\mathbf{R}$, we make use of the functions `toeplitz` and `tril`. The functions `toeplitz` and `tril` are often found in software used for digital signal processing. The operation of these functions is illustrated in Appendix C. The algorithm to construct $\mathbf{R}$ is as follows:

Step 1: Let

$$\mathbf{c} = \begin{bmatrix} c_{i,j}[N-1] & \cdots & c_{i,j}[-N+1] \end{bmatrix}$$

where

$$c_{i,j}[m] = \sum_{k=-\infty}^{\infty} c_i^*[k] c_j[k+m].$$

[7] The sparsity of a matrix is measured by the number of elements which are zero. Therefore, a sparse matrix has many zero elements.

Step 2: Append $\mathbf{c}$ with $2(L-1)N+1$ zeros.

Step 3: Calculate $\mathrm{tril}[\mathrm{toeplitz}(\mathbf{c})]$.

Step 4: Omit the first $N-1$ rows and the last $N-1$ columns, and keep rows $1, N, \ldots, 2LN+1$. This yields $\mathbf{R}_{i,j}$.

Step 5: Construct

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_{1,1} \\ \vdots \\ \mathbf{R}_{M,M} \end{bmatrix}.$$

An example of the algorithm which illustrates its correctness is given in Appendix C. This algorithm can take advantage of highly optimised implementations of `toeplitz` and `tril`. Furthermore, Step 4 can be vectorised or optimised memory-wise. Therefore, the algorithm allows for fast reconstruction of $\mathbf{R}$.

8.5 Conclusion

We have discussed an algorithm to reconstruct in real-time the power spectral density, or equivalently the autocorrelation, of a signal sampled at sub-Nyquist frequencies. We notice that $\mathbf{R}$ and thus $\mathbf{R}^\dagger$ can be precomputed, and all matrix manipulations experience a significant speedup because $\mathbf{R}$ is usually sparse. Therefore, our algorithm allows for fast real-time estimation of the power spectral density of the sampled signal. This is in contrast to other methods available, since these methods are mostly based on optimisation or complex algorithms, which yields computationally expensive methods.

9 | Sampling methods

9.1 Introduction

This chapter will describe different sampling techniques that make up the sampling component described in Section 6.3. These sampling techniques will be developed according to their specifications, which were given in Section 6.2.

A lot of theory on different sampling methods has already been published. For example, [2], [7], [8] describe different methods and describe how to implement those methods in their respective frameworks. Each of these methods impose their own restrictions on the design of the sampling device. *Circular sparse sampling* [2] describes a technique in which a device uses multiple samplers which all sample at the same frequency. *Collaborative sampling* [7] describes a technique in which multiple devices use multiple samplers. All these samplers sample at the same frequency. *Coprime sampling* describes a technique in which exactly two devices are used, but whose sampling frequencies differ[1].

In this chapter we further investigate *circular sparse sampling*, *collaborative sampling* and *coprime sampling*. We will fit them into our system in such a way that they can collaborate with the reconstruction algorithm, eventually allowing for correct operation of the detection.

9.2 Concept

In this section we will analyse the different sampling techniques described in the introduction and connect them to our system. Before we move on, the reader must understand the difference between a sampler and a device. A device refers to a physical device, which may employ multiple samplers. A sampler is part of a device, and performs actual sampling of the signal. Remember that the term coset and sampler are used interchangeably. Also keep in mind that everything will be explained conceptually.

Circular sparse sampling

Circular sparse sampling is a form of multi-coset sampling in which samplers all have the same sampling period NT, where T is the sampling period of the in-

[1] More specifically, their sampling frequencies are coprime.

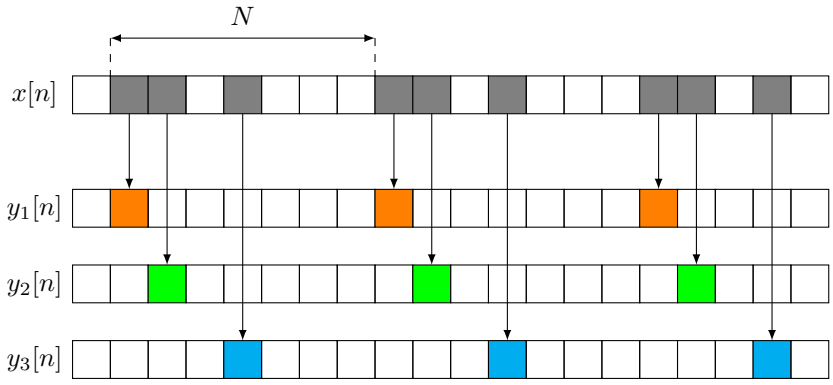


Figure 9.1: An example of circular sparse ruler sampling with three samplers and $N = 7$

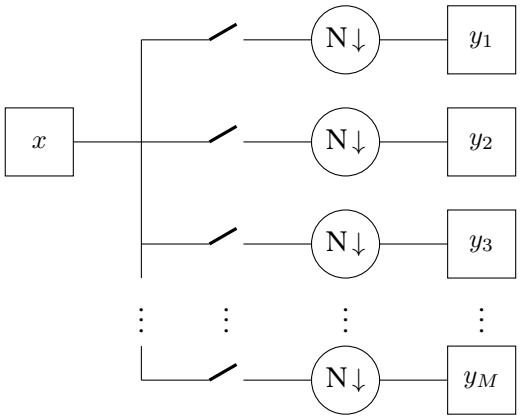


Figure 9.2: Conceptual schematic of a circular sparse ruler implementation. There may be an offset between the different decimations.

put signal $x[n]$ sampled at the Nyquist frequency and N the downsampling factor. Here N refers to the downsampling factor discussed in Section 8.3. An example of circular sparse sampling where $N = 7$ is illustrated in Figure 9.1.

The grey blocks represent the samples recovered by all samplers of the device. The coloured blocks represent the samples that are recovered by the different samplers. A possible conceptual implementation of this sampling method is illustrated in Figure 9.2. Here switches represent a sample operation of the input signal x. These switches sample simultaneously at the Nyquist frequency. The N$\downarrow$ represent an N-decimation. As explained in Section 8.3, there may be an offset between the decimations of the different cosets. Therefore, the output of every coset is an N-decimation of the input signal, but there may be an offset in time between the different outputs. This is depicted in Figure 9.1. Note that this is similar to the description of multi-coset sampling in Section 8.3.

Figure 9.2 shows that the input signal is sampled at the Nyquist frequency and then decimated. It is important to notice that this operation is equivalent to sampling at

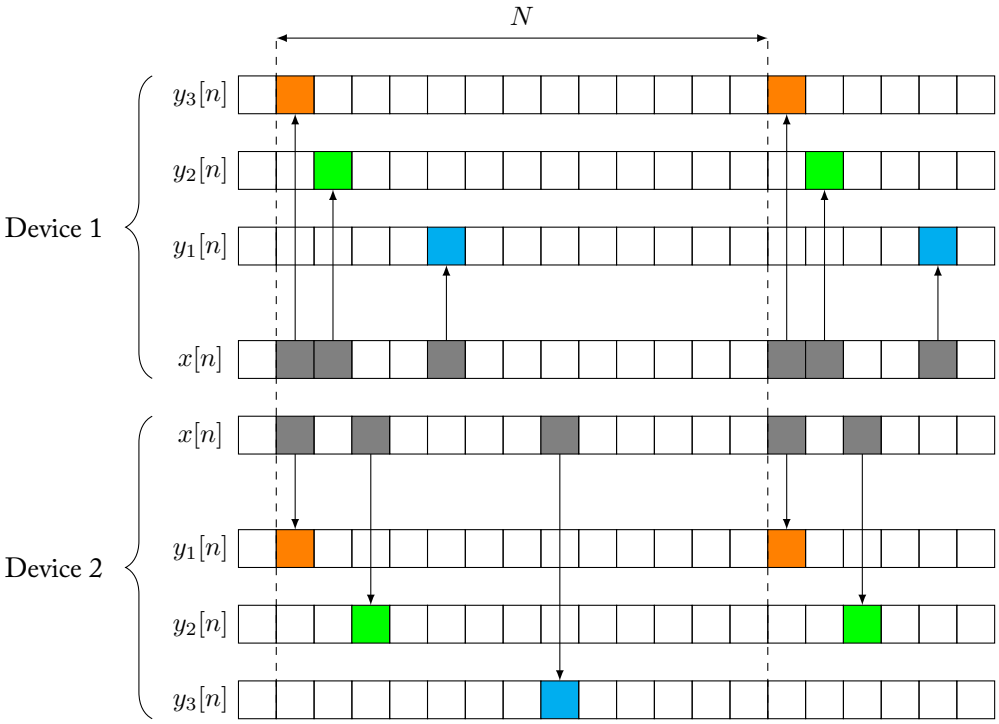


Figure 9.3: Collaborative sampling with two devices with each three samplers

a frequency N times lower than the Nyquist frequency. Therefore, the outputs of the cosets can be obtained by sampling the input signal at sub-Nyquist frequencies, which is what we aim to do. In practice it is implemented this way. For conceptual purposes however, we decide to stick to the representation shown in Figure 9.2.

Collaborative sampling

Collaborative sampling is similar to circular sparse sampling, except for the fact that circular sparse sampling is restricted to only one device, while collaborative sampling allows for multiple devices. Collaborative sampling is illustrated in Figure 9.3. In this example, we use two devices with three samplers each and a downsampling factor of $N = 13$.

In Figure 9.3, the top half represents the first device, and the bottom half represents the second device. Both devices sample separate signals. Similar to Figure 9.1, the coloured blocks represent the samples recovered by the sampler of the associated device. A conceptual schematic of a possible implementation is given in Figure 9.4.

Coprime sampling

Coprime sampling is different from circular sparse sampling and collaborative sampling in the sense that the samplers sample with different periods. Also, the amount

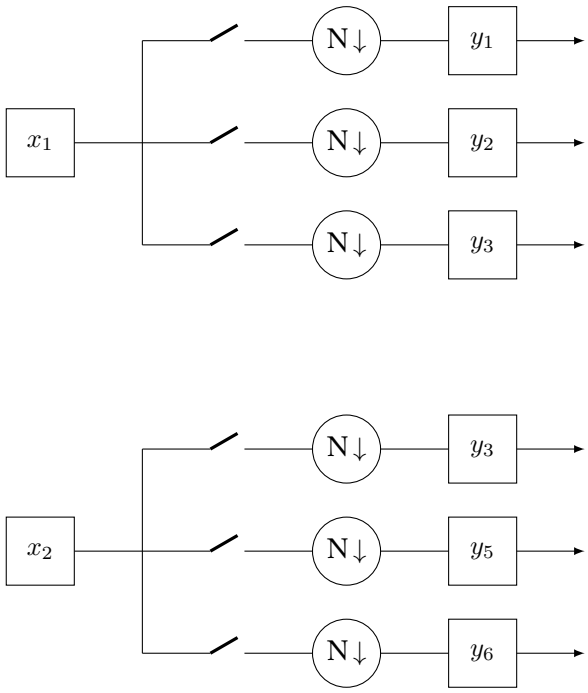


Figure 9.4: Conceptual schematic of collaborative sampling implementation. There may be an offset between the different decimations.

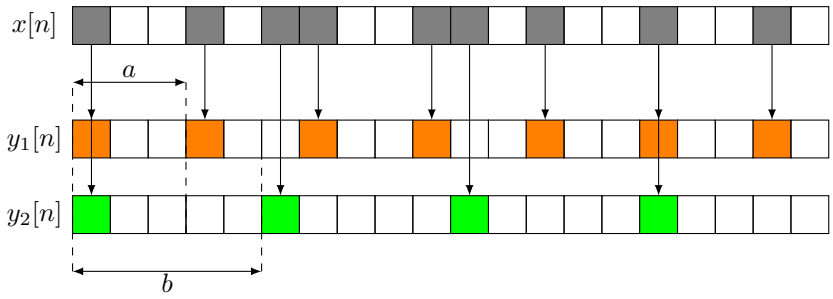


Figure 9.5: Coprime sampling with $a = 3$ and $b = 5$

of samplers is restricted to two. The first sampler samples with period aT, and the second sampler samples with period bT. Coprime sampling requires that a and b are coprime [8]. The technique is illustrated in Figure 9.5. In this example, $a = 3$ and $b = 5$. The choice of the a and b substantially influences the performance of the coprime sampler. This influence will be investigated later in this chapter.

In Section 8.3, it is assumed that every sampler samples the input signal at the same sampling rate. However, coprime sampling involves two samplers sampling at different rates. Therefore, using the reconstruction algorithm in conjunction with coprime sampling is not trivial. Figure 9.6 shows an equivalent hardware implementation, which may be used for reconstruction. This implementation will be further discussed later on. Note that the so called equivalent samplers u_i and v_i

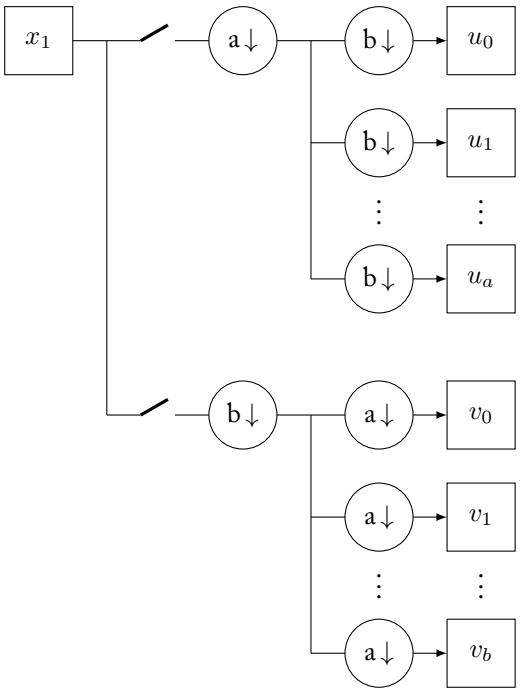


Figure 9.6: Conceptual schematic of an equivalent implementation of coprime sampling. Although coprime sampling is technically not regarded as a form of multi-coset sampling, the equivalence of a multi-coset implementation allows us to do so. There may be an offset between the different decimations.

yield ab-decimations of the input signal.

9.3 Sampling and reconstruction

In this previous section we discussed three multi-coset sampling methods. In Section 8.3, we saw that the sampling method determines the configuration, which is the way the samplers sample the input signal. The configuration then yields $c_i[n]$. Remember that the sampling signal $c_i[n]$ represents the way sampler i samples the input signal. If these $c_i[n]$ satisfy the requirements stated in Section 8.4, then the algorithm described in Section 8.3 can be used to reconstruct the power spectral density of the sampled signal. We will reconsider these requirements one by one.

First, every $c_i[n] = 0$ for $n < 0$ and $n > N - 1$. Since $c_i[n] = 1$ if coset i samples the $N - n$'th sample of every group of N samples of the input signal, which was discussed in Section 8.3, this requirement is satisfied.

Second, all $c_i[n] = 1$ for a single $n = n_i$ and otherwise zero. Every coset i samples a single sample of every group of N samples of the input signal. Therefore, this requirement is also satisfied. It is very important to notice this requirement in conjunction with the first requirement tells us that n_i determine the offsets of the decimations of the different cosets.

Finally, the circular sparse ruler problem, stated below, must be satisfied.

Circular sparse ruler problem Consider n_i for every coset i as described in the second requirement. The set consisting of $|n_i - n_j|$ and $N - |n_i - n_j|$ for all combinations of cosets i and j must make up $0, \ldots, N-1$.

Let $d = |n_i - n_j|$ or $d = N - |n_i - n_j|$ for a combination of cosets i and j. Equivalently, d is the integer difference between the offsets of the decimations of cosets i and j. We now say that *lag d is estimated*. We will use this terminology throughout the chapter. The circular sparse ruler problem can be reformulated as follows.

Circular sparse ruler problem (alternative formulation) Lags $0, \ldots, N-1$ must be estimated.

We have abstracted the satisfaction of the requirements necessary for reconstruction to the alternative formulation of the circular sparse ruler problem. The circular sparse ruler problem is treated in a more abstract way in Appendix D. Also, we succeeded in formulating the problem as an integer linear program. This allowed us to calculate optimal solutions. A detailed derivation of the formulation of the integer linear program can be found in Appendix E. From now on, we assume that we have solutions to the circular sparse ruler problem readily available.

Circular sparse ruler sampling

Since the solutions to the circular sparse ruler problem are readily available, we can choose all $c_i[n]$ such that the problem is satisfied. Then all requirements stated in Section 8.4 are satisfied and the sampled signal can be reconstructed. It is important to note that $c_i[n]$ is defined for $i = 1, \ldots, M$, which was explained in Section 8.3. Therefore, the solution to the circular sparse ruler problem yields a value for M. The parameter M determines the amount of samplers which are required. Since we want to minimise the amount of samplers, we should choose a solution to the circular sparse ruler problem such that M is minimal. This is discussed in Appendix D.

Collaborative sampling

In collaborative sampling, the estimation of the lags is split across two physically separate devices. This situation is analysed and described more carefully in Appendix D. The advantage of splitting the estimation of the lags across physically separate devices is that this reduces the workload of a single device.

To ensure that reconstruction is possible, we consider a treatment similar to Section 9.3. However, the difference is that Step 6 of the reconstruction algorithm discussed in Chapter 8 is invalid, because a single device does not satisfy the circular sparse ruler problem. However, since a single device does estimate certain lags, an analysis similar Appendix B shows that each device solves its own part of

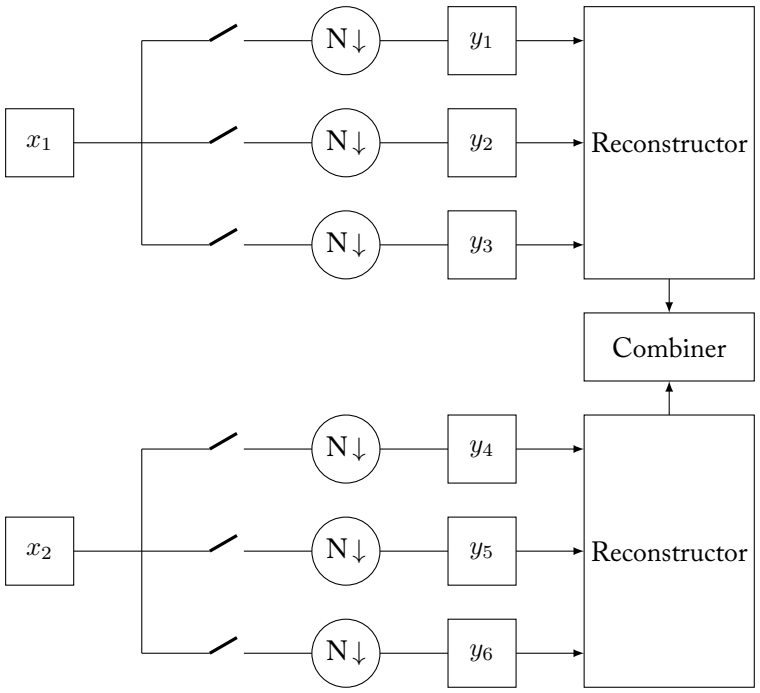


Figure 9.7: Conceptual schematic of collaborative sampling implementation with a combiner. There may be an offset between the different decimations.

the equation of Step 6 in such a way that their results yield the complete solution. Therefore, after the reconstruction of each device, it remains to combine their results. A possible final implementation is pictured in Figure 9.7.

Coprime sampling

As we argued before, the algorithm design in Chapter 8 is not designed to process two signals sampled by coprime sampling. This yields that some preprocessing is necessary. The idea is to break the signal sampled by coprime sampling into many different signals which seem to be sampled uniformly. More specifically, we break the signals of the two different samplers with sampling periods aT and bT into artificial signals with sampling period abT. This is illustrated in Figures 9.6 and 9.8.

In Figure 9.8, the grey blocks represent the samples recovered by the coprime sampling device. The orange blocks represent the samples that are recovered by the sampler with sampling period aT, and the green blocks represent the samples that are recovered by the sampler with sampling period bT. Finally, the bottom block shows how the orange and greens blocks are split across many artificial signals. These artificial signals represent samplers with an uniform sampling period. Therefore, these artificial samplers make reconstruction possible. Coprime sampling is discussed more carefully in Appendix D.

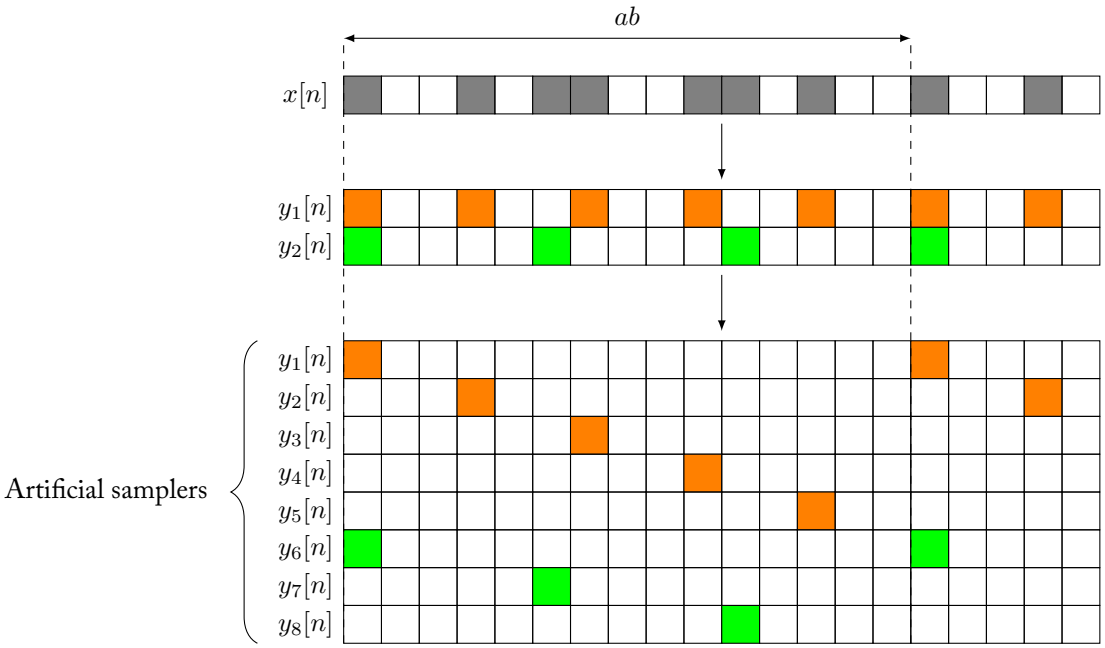


Figure 9.8: Preproccesing of a signal sampled by coprime sampling

9.4 Conclusion

We have discussed three sampling techniques that can be used with our system. Each method imposes different constraints and offers different advantages such that the specifications in Section 6.2 are satisfied. That is, circular sparse sampling and coprime sampling can be used for sampling with a single device, whereas collaborative sampling can be used for sampling with multiple devices. All sampling techniques, however, allow for correct operation of the detection.

10 | Detection

10.1 Introduction

In the previous chapters we have discussed how an estimate of the autocorrelation, or equivalently the power spectral density, of the input signal can be constructed. This estimate will serve as input for the detector.

This chapter introduces several methods to perform the detection process as described in Section 6.3. It is worthwhile to note that unlike conventional detectors, our detector has no access to the original received signal $x[n]$. Furthermore note that the estimate of the autocorrelation, $r_x[m]$, is the autocorrelation of a *wide-band* signal. It therefore possibly conveys information of multiple frequency bands. As the detector has to distinguish between noise and signal on a frequency basis (Section 6.2), this affects the detection methods that can be used in our system.

We will start this chapter with a global overview of detection methods available. Based on this overview we will motivate the analysis of two detection methods, which are described later on. This analysis will lead to a conclusion of this chapter. For the technical reader a more in depth analysis is included in Appendix F.

10.2 Overview

The fundamental problem of detection is to discern noise from a signal. This problem can be formulated as a binary hypothesis test. Given a signal $x[n]$ a detector has to decide between the hypotheses

$$\begin{aligned} \mathcal{H}_0 : &\quad x[n] = w[n], \\ \mathcal{H}_1 : &\quad x[n] = w[n] + s[n] \end{aligned}$$

where $w[n]$ denotes a noise signal and $s[n]$ a signal different from noise. Under $\mathcal{H}_0$ the signal $x[n]$ contains only noise. Under $\mathcal{H}_1$ another signal besides noise is present in $x[n]$. Besides this hypothesis test, the detector also has to deal with frequency bands: which frequency bands are occupied by some signal and which are unoccupied?

We can distinguish between two types of detection methods that provide an answer to this problem:

1. The first type of methods are detection methods based on information of the type of signals to be detected, such called *prior information*. A popular example of this type is cyclostationary detection which uses the knowledge of the modulation techniques of the signals to perform the detection [12], [20]. Another example is a matched filter detector. This detector correlates the received signal with a known signal that is to be detected and therefore needs exact knowledge of the signal [10], [21].

2. The second type of methods are detection methods methods working without any prior knowledge of the signals to be detected, such called *blind* detection methods. A basic detection method of this type is energy detection [12] which compares the energy contained in a signal to a certain threshold based on the noise power. Other examples include Covariance Absolute Value detection [14] and wavelet-based detection [13].

A well known problem in the context of cognitive radio is the *SNR wall*. This problem states that below a certain signal-to-noise ratio[1] a detector fails to detect a signal no matter how long the sensing time [22]. This problem arises from uncertainties and imperfections in the model assumptions of that detector. Detectors that use a threshold that is independent from noise or signal parameters are not affected by this SNR wall [12] and possess the property of constant false-alarm rate (*CFAR*). This property states the probability that the detector decides that its input signal is a signal different from noise, where in fact the signal contains only noise, is constant.

In this chapter we will focus on detection methods that do not assume to have any prior knowledge available. Although methods that use *prior information* can operate at lower SNRs than blind detectors, they generally result in more (computationally) complex algorithms. Furthermore, using a detector that depends on *prior information* cannot be used to detect arbitrary signals as demanded in Section 6.2. [2]

This chapter will be concerned with the theoretical analysis of the following two detection methods:

1. Energy detection. This detection method uses a test statistic which is easy to compute. Its threshold, as stated before, is dependent on the noise power. This noise power has to be estimated in real detectors for example by using an empty frequency band as reference. This detection method is therefore subject to the SNR-wall phenomenon as there will be uncertainty in the noise estimate.

2. Covariance Absolute Value detection (CAV). This detection method uses a test statistic based on the autocorrelation of the signal. It uses a threshold that

[1] The signal-to-noise ratio is the ratio of the signal power and the noise power.

[2] We will not focus on spread-spectrum modulation in this thesis. Spread spectrum signals occupy a wide band of the spectrum and are designed to be resilient against interception by other detectors than those of the intended receivers [23], [24].

does not depend on any signal or noise parameters. Therefore, this detector does posses the *CFAR* property [25].

This analysis represents the search for a suitable detector to be used in our system. The analysis of energy detection is motivated by the relative simplicity of the algorithm and low complexity of its implementation. Analysing CAV is motivated by the fact that it is a *CFAR* detector which is convenient in practical applications [12]. Based on this analysis this chapter will conclude why the final system is using an energy detector.

10.3 Energy detection

Conventional energy detection

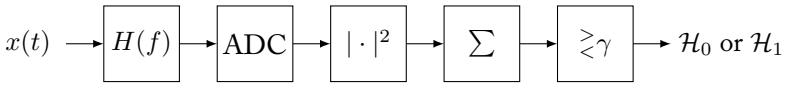


Figure 10.1: Conventional energy detector

A standard digital energy detector[26] is a detector that consists of a low-pass filter $H(f)$ that filters the analog input signal $x(t)$ such that only the frequency band of interest remains[3]. This filtered signal is then converted to a digital signal $x[n]$ by an analog-to-digital converter (ADC). Then the detector takes the square of the absolute value of each sample. The detector then estimates the energy Λ of the signal during a period of N samples by summing N of those samples as

$$\Lambda = \sum_{n=0}^{N-1} |x[n]|^2. \tag{10.1}$$

This estimate is then compared to a threshold γ to decide whether a signal other than noise is present in $x[n]$. That is, Λ serves as test statistic for the binary hypothesis problem

$$\begin{aligned} \mathcal{H}_0 :&\quad \text{if } \Lambda < \gamma, \\ \mathcal{H}_1 :&\quad \text{if } \Lambda > \gamma. \end{aligned}$$

The details behind determining the threshold γ and a derivation of this energy detector can be found in Appendix F.2.

Some energy detectors use Λ/N as test statistic [27], which is an estimate of the average power instead of the energy. Using this test statistic, a detector that uses the autocorrelation $r_x[m]$ can be constructed. Such a detector is depicted in Figure 10.2.

[3]By using a mixer, a lowpass filter can always be used to for this purpose.

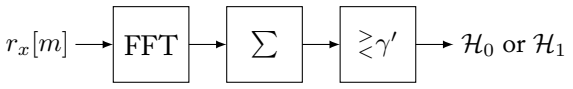


Figure 10.2: Energy detector using the power spectral density. Here $r_x[m]$ is the autocorrelation of a filtered signal to the band of interest.

This detector calculates a discrete estimate of power spectral density by applying the Fast Fourier Transform to $r_x[m]$. By summation over the obtained power spectral density an estimate of the average power of $x[n]$ is obtained. Finally, the result is compared to a threshold γ' to decide between $\mathcal{H}_0$ and $\mathcal{H}_1$. Here $\gamma' = \gamma/N$.

Energy detection with knowledge of the reconstructor

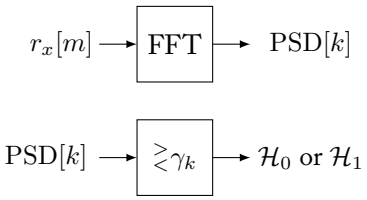


Figure 10.3: Energy detection with knowledge of the reconstructor. Here $r_x[m]$ is the autocorrelation of a wideband signal.

The problem with the conventional energy detectors as described above is that they assume that the signal is filtered to only contain the frequency band of interest. Filtering the input signal, however, affects the reconstruction process of $r_x[m]$: by prefiltering the input signal before reconstruction, the reconstructed autocorrelation will correspond to a small band signal (the band of interest) instead of a wideband signal. We can therefore not prefilter the signal.

A modified energy detector as proposed in [2] is depicted in Figure 10.3. The detector first estimates the power spectral density by applying the Fast Fourier Transform to an estimate of the autocorrelation function. This results in a discrete estimate of the power spectral density. Each element of this estimate is then compared to an element-dependent threshold γ_k. As this detector detects per frequency whether there is a signal present or not, filtering of the signal is not necessary anymore. To calculate γ_k information about the reconstruction method as described in Chapter 8 is used. Details on this calculation can be found in Appendix F.3.

Noise estimation

As energy detectors rely on knowledge of the noise power it is necessary to have an estimate of this noise power. This can be done, for example, by taking the mean power level in an empty frequency band as estimate. If such a band is not known the noise power can be estimated by using a sliding window over the estimated power spectral density. The lowest mean observed power in this window is then taken as an estimate of the noise power.

10.4 Covariance absolute value method

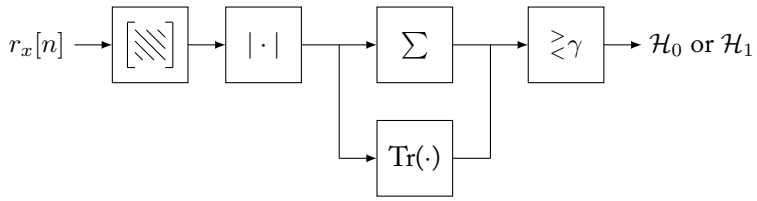


Figure 10.4: CAV detector

The Covariance Absolute Value (CAV) detector was first introduced in [14]. A block diagram of a CAV detector is depicted in Figure 10.4, from which it can be seen that it performs 5 steps:

Step 1: Calculate the $L \times L$ symmetric toeplitz matrix $\mathbf{C}$ using the first L samples of $r_x[n]$ as the first column. The matrix $\mathbf{C}$ is given by

$$\mathbf{C} = \begin{bmatrix} r_x[0] & r_x[1] & \dots & r_x[L-1] \\ r_x[1] & r_x[0] & \dots & r_x[L-2] \\ \vdots & \vdots & \ddots & \vdots \\ r_x[L-1] & r_x[L-2] & \dots & r_x[0] \end{bmatrix}. \tag{10.2}$$

Step 2: Take the absolute value of all elements of $\mathbf{C}$.

Step 3: Calculate the sum of all elements, that is $T_1 = \sum_{n=1}^{L-1} \sum_{m=1}^{L-1} C_{nm}$.

Step 4: Calculate the trace of $\mathbf{C}$, that is $T_2 = \text{Tr}(\mathbf{C})$.

Step 5: Compare T_1/T_2 to a threshold γ. In case that $T_1/T_2 > \gamma$ it is decided that $\mathcal{H}_1$ is true, otherwise $\mathcal{H}_0$ is regarded as the true hypothesis.

A more technical explanation of CAV can be found in Appendix F.4. From this technical analysis it becomes clear that the threshold as given in [14] cannot be used directly with our reconstruction method and will therefore need a re-derivation.[4]

Filtering

To be able to detect in a certain frequency band the input signal should be filtered by a band-pass filter. However if the input signal $x[n]$ is filtered, then the noise in that input signal is also filtered. This affects the autocorrelation function $r_x[m]$ and therefore affects the test statistic T_1/T_2 of the CAV detector. A technique to circumvent this problem is proposed in [14], but requires that the filter applied to $x[n]$ is known. However, we start our detection process with $r_x[m]$ instead of $x[n]$

[4]The elements of the autocorrelation as estimated by our reconstructor exhibit a variance that is different from the elements of the estimate as assumed in the derivation in [14].

and moreover we cannot filter the input signal of the reconstructor. Therefore this technique cannot be applied. This limits the CAV implementation as described in this section to detection in just one frequency band. Furthermore, if use of this technique was possible, it would imply that we need to apply it per frequency band which may significantly add up to the computational cost in case that wide-band spectra are used.

10.5 Conclusion

In this chapter we have presented two different detection methods. We have observed that using our reconstruction method as described in Chapter 8 affects the applicability of the default implementations of these detectors. Furthermore we have noticed that the energy detection method for use with our reconstruction method is affected by the SNR wall. Although the CAV detection method does not suffer from this SNR wall due to its *CFAR* property, it does introduce a new problem when it is required to detect in multiple frequency bands. Moreover the default threshold as derived in [14] needs to be re-derived if CAV detection is to be used with our reconstruction method. Based on this analysis, we therefore conclude that the modified energy detector is most suitable for use in combination with our reconstruction method: energy detection has a relatively low-computational cost and can be used to detect in multiple frequency bands. Using CAV detection needs additional analysis to support multiple frequency bands.

11 System evaluation

11.1 Introduction

Chapters 7 to 10 have analysed and described several sampling, reconstruction and detection techniques. It is in this chapter that we will evaluate their performance. A discussion will be based on the specifications as described Section 6.2. The chapter will start with an overview of the complete system, to continue with a description of the tests performed to assess the performance of the system. We will end with the results of those tests. These results will be further discussed in Chapter 12.

11.2 Overview

A block diagram of the complete system is depicted in Figure 11.1. Considered options that did not make it into the final system are grayed out.

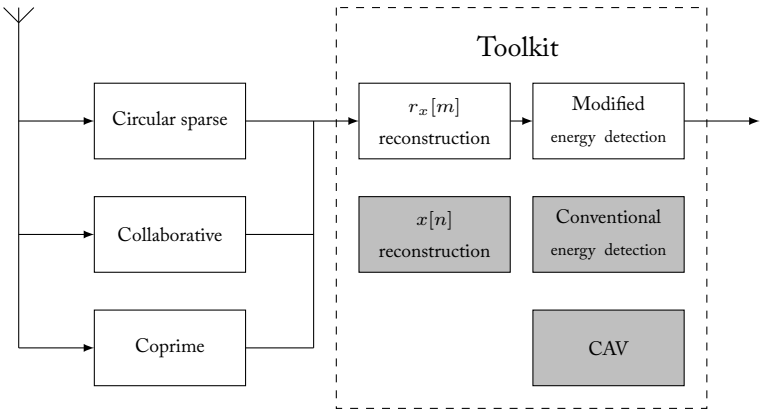


Figure 11.1: System overview with design choices

11.3 Testing

To evaluate the performance of our system we will we use an artificial testing signal to test both the reconstructor and the detector. Details on the generation of this signal can be found the subsequent section.

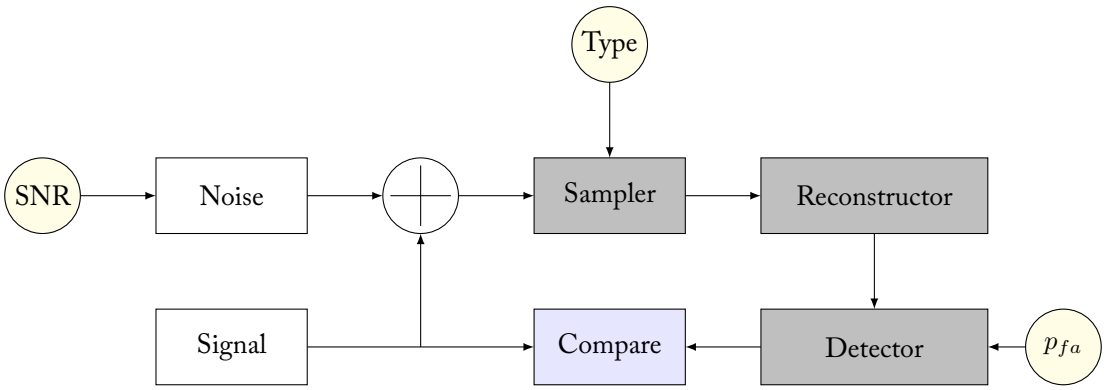


Figure 11.2: Testing the system

Test signal

The test signal will consist of an ideal signal polluted with noise. The signal power and noise power will be based on a specified signal-to-noise ratio. Given a signal-to-noise ratio, we can construct the testing signal by doing as follows:

1. Generate the ideal signal $s[n]$. The signal will consist of filtered circular symmetric complex Gaussian noise such that it represents a bandpass signal with $-0.2 \leq f \leq 0.2$, where f denotes the usual relative frequency.
2. Generate the noise signal $w[n]$, which will be additive circular symmetric complex Gaussian noise. The average power of the signal and the noise are chosen such that the required SNR is met.
3. Add $s[n]$ to $w[n]$ to produce the testing signal $x[n]$.

Tests

There are three parameters of the system, represented by circles in Figure 11.2, that will be used in our tests to assess the performance of the system. The parameters are as follows:

1. The SNR of the input signal.
2. The sampler type used to sample the input signal.
3. The false alarm probability p_{fa} of the detector.

Our tests will be targeted at

1. addressing the performance of the reconstruction by feeding it an artificial constructed signal while
    a) varying the sampling technique used to sample the signal, described in Chapter 9 and

b) varying the compression rate and the sampling time, described in Chapter 8.

By comparing the reconstructed power spectral density to the power spectral density recovered directly from the constructed test signal, we can evaluate the correctness of reconstruction when used with a specific sampling technique.

2. addressing the performance of detection on the reconstructed signal. The modified energy detector as described in Section 10.3 will be evaluated by tests that make use of the test signal as described in Section 11.3. Detection is applied to the output of the reconstructor while

   a) varying the signal-to-noise ratio of the test signal and

   b) varying the false-alarm probability.

   The sampling technique is fixed to circular sparse sampling. The reason for this is that the dependence of the detector's performance on the reconstructor's performance, is such that techniques with equivalent reconstruction performance yield approximately equivalent detection performance. Therefore, we can use the results of the reconstructor to extrapolate the results of the detector.

   By extensive simulation we can construct estimates of the *receiver operating characteristic curves*, which plot the detection probability versus the false alarm probability of the detector. From these curves the minimum signal strength and the detectors performance on a random signal can be evaluated. Furthermore the detection performance of the system as a whole can be evaluated by comparing these curves.

11.4 Results

The performance of the reconstructor is depicted in Figures 11.3a to 11.3c, which are accompanied by Tables 11.1 to 11.3. The performance of the detector is depicted in Figures 11.4a, 11.4b, 11.5a and 11.5b. Their results will be evaluated separately. Before we continue, the reader must be aware of the following two concepts.

First, we argued in the introduction the we want to avoid sampling at the Nyquist frequency. It turned out that multi-coset sampling offered a solution to this problem. To evaluate the different sampling techniques, we introduce a quantity called *compression*. Compression is the percentage of the number of samples the sampling technique requires with respect to the samples obtained when sampling at the Nyquist frequency. Therefore, high compression represents an effective sampling technique.

Second, we introduce the normalised mean squared error (NMSE) to assess the performance of the reconstructed power spectral density.[1] A lower NMSE represents a better reconstruction of the power spectral density.

[1] If $\hat{\text{PSD}}$ represents the estimated power spectral density, then the NMSE is given by $||\text{PSD} - \hat{\text{PSD}}||/||\text{PSD}||$.

Sampling technique	**Compression**	**Configuration**	**Downsampling**
Circular sparse ruler sampling	69%	4 samplers	$13\times$
Coprime sampling	55%	2 samplers, periods $4T$ and $5T$	$4\times$
Collaborative	54%	6 samplers over 2 devices	$13\times$

Table 11.1: Description of the sampling techniques depicted in Figure 11.3c

Reconstruction

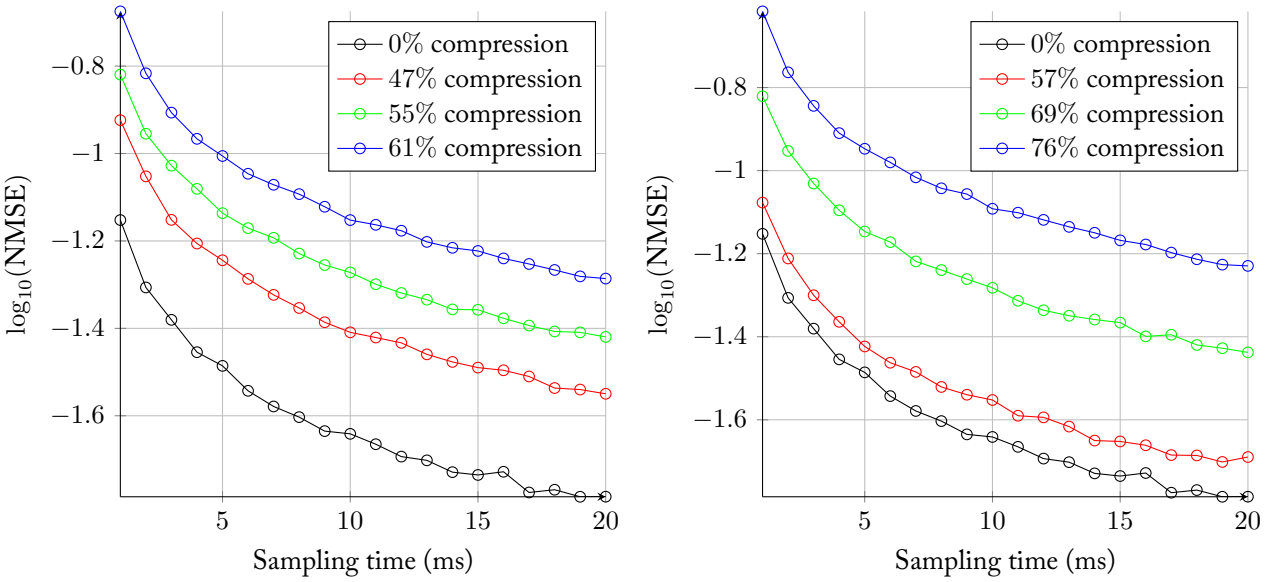
The performance of the reconstruction will by measured by varying the sampling time, where we illustratively assume a Nyquist frequency of $25\,\mathrm{Mhz}$, and by varying the sampling technique. These sampling techniques can all achieve different compressions.

Figure 11.3a shows the NSME for coprime sampling. We see that the error increases as the compression increases, but decreases as the sampling time increases. A similar observation applies to Figure 11.3b. Also, we see that for a similar compression, circular sparse ruler provides less error than coprime sampling. The precise configuration is described in Tables 11.2 and 11.3. These tables show us that circular sparse ruler allows for more downsampling, but also requires more samplers. Note that the resolutions of the reconstructed spectra differ. This should be taken into account when comparing the two methods. Figure 11.3c shows three sampling techniques which have equivalent performance. We observe that their performance gain by increasing the sampling time is equivalent. Table 11.1 shows that in the case of equivalent performance, coprime sampling requires the least number of samplers. Also, in that case, coprime sampling has the least downsampling and circular sparse ruler sampling has the highest compression.

Detection

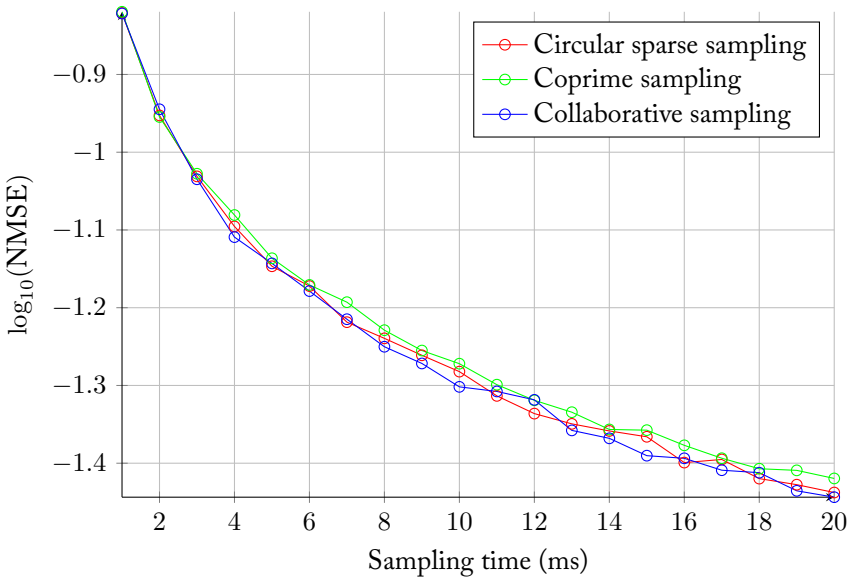
Figures 11.5a and 11.5b show the performance of the detector in the case that circular sparse ruler sampling with 69% compression is used. The performance is shown for signals with $-5\,\mathrm{dB}$, $0\,\mathrm{dB}$ and $5\,\mathrm{dB}$ SNR. In addition to detection using the theoretical noise level, we have simulated uncertainty in noise power by adding $0.5\,\mathrm{dB}$ to the theoretical noise level.

The 802.22 IEEE standard tells us that a detection probability of 0.9 can be considered sufficient. We see that in case that the noise power is known, signals with approximately $0\,\mathrm{dB}$ SNR or higher can be detected sufficiently. However, in the case that the noise power is not accurately known, a detection probability of approximately 0.9 requires a relatively high false-alarm probability. Finally, if we increase the compression of the used sampling technique, the detection performance deteriorates.



(a) Coprime sampling. The curves are described in Table 11.3.

(b) Circular sparse sampling. The curves are described in Table 11.2.



(c) Equivalent performance of different techniques. The curves are described in Table 11.1.

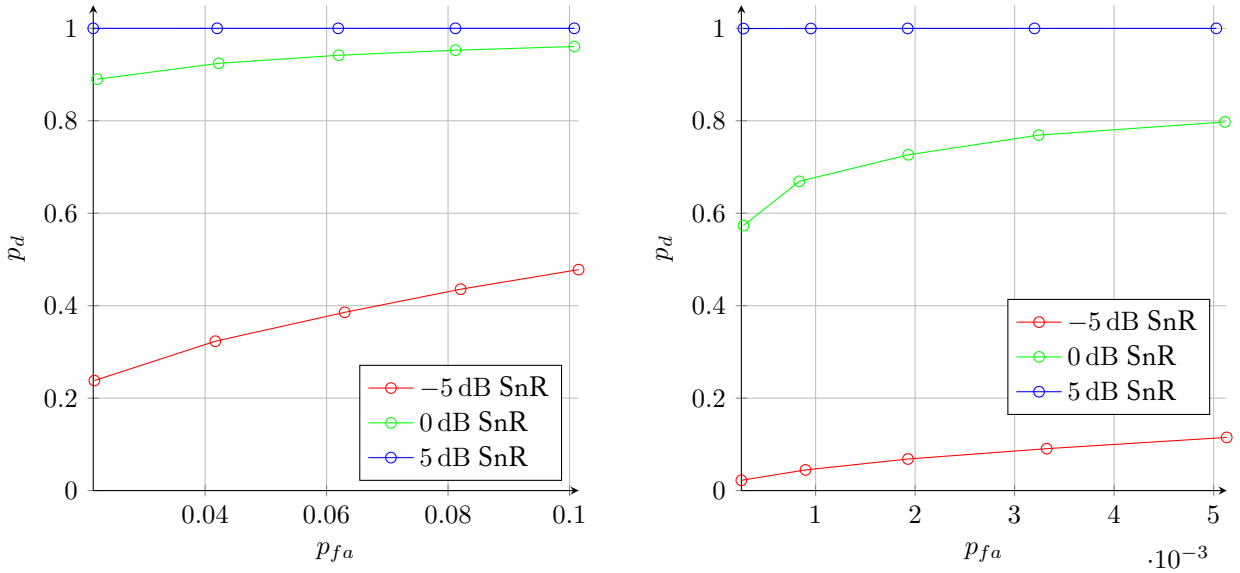
Figure 11.3: Normalised mean squared errors of the reconstructed power spectral densities by different sampling methods. The plots assume a Nyquist frequency of 25 MHz. The Nyquist sampled signal is limited in support such that $r_x[m] = 0$ for $|m| > 60$. Note that the different techniques yield different resolutions.

Sampling technique	Compression	Configuration	Downsampling
Circular sparse ruler sampling	57%	3 samplers	7×
Circular sparse ruler sampling	69%	4 samplers	13×
Circular sparse ruler sampling	76%	5 samplers	21×

Table 11.2: Description of the sampling techniques depicted in Figure 11.3b

Sampling technique	Compression	Configuration	Downsampling
Coprime sampling	47%	2 samplers, periods $3T$ and $5T$	3×
Coprime sampling	55%	2 samplers, periods $4T$ and $5T$	4×
Coprime sampling	61%	2 samplers, periods $4T$ and $7T$	4×

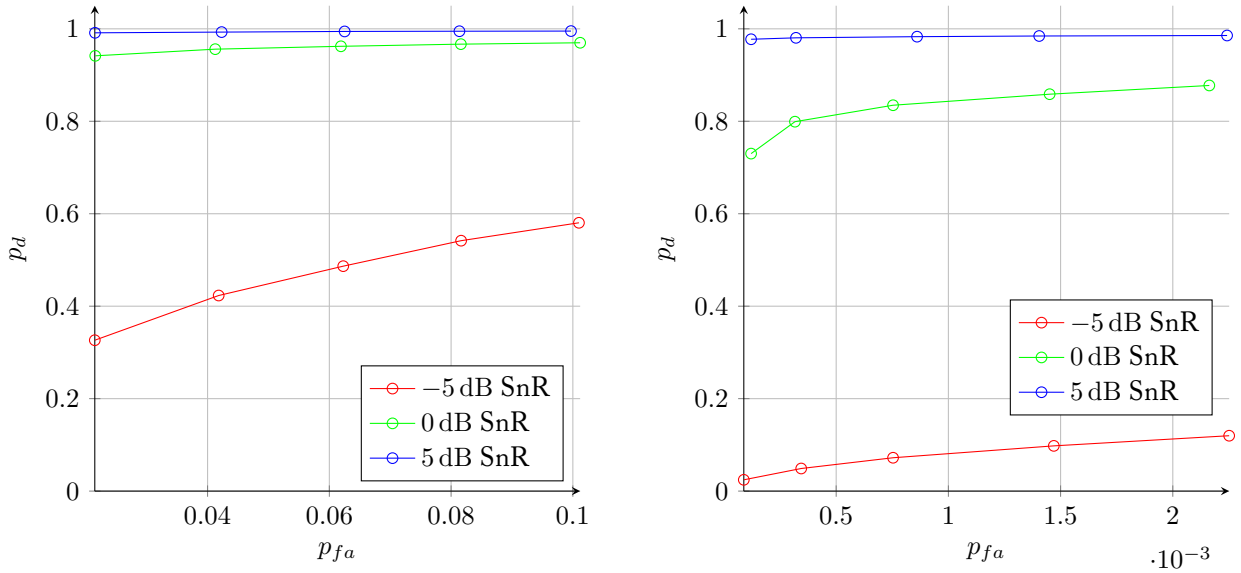
Table 11.3: Description of the sampling techniques depicted in Figure 11.3a



(a) The case of no noise power uncertainty

(b) The case of 0.1 dB noise power uncertainty

Figure 11.4: The detector's receiver operation characteristic. Circular sparse sampling with 76% compression is used.



(a) The case of no noise power uncertainty

(b) The case of 0.1 dB noise power uncertainty

Figure 11.5: The detector's receiver operation characteristic. Circular sparse sampling with 69% compression is used.

12 | Discussion

12.1 Introduction

This chapter will reflect on the results which were presented in Chapter 11. We will carefully reconsider the specifications stated in Chapter 6.

12.2 Sampling

Recall that the techniques used for sampling *must* allow for

1. correct operation of the detection;
2. usage of a single sampling device and
3. usage of multiple sampling devices such that the workload can be distributed across the devices.

In addition, the sampling techniques *should* allow for

1. sampling as efficient as possible.

We have designed three sampling techniques to fit our system. We saw that together, these techniques met their requirements. First, all techniques where refined to enable correct operation of the reconstructor. Second, coprime sampling and circular sparse sampling allow for the usage of one device, and collaborative sampling allows for the multiple devices. Finally, we have abstracted the correct operation of the reconstruction to the satisfaction of the circular sparse ruler problem. Since the solution of this problem influences the efficiency of the sampling technique, we can optimise to find solutions such that sampling techniques are as efficient as possible. This optimisation is discussed in Appendix E.

A comparison between the different sampling techniques has shown that each sampling technique has its own advantages. For example, circular sparse ruler sampling allows for the biggest compression, coprime sampling for the least amount of samplers and collaborative sampling for the usage of multiple devices while distributing the workload. We can therefore conclude that the sampling technique must be chosen according to the situation, and there is no 'one size which fits them all'.

In addition to meeting the requirements, we did some research on the circular sparse ruler problem to further optimise finding solutions. This was discussed in Appendix D. The circular sparse ruler problem allowed us to construct solutions for multi-coset sampling with comparably higher compression than the solutions proposed in [2].

We can also assess the similarities of the different sampling techniques. For example, circular sparse sampling can be considered a form of collaborative sampling with one device. This observation arises the question if is possible to generalise all sampling techniques to an universal technique. This universal technique could then be optimised. This is considered future work.

12.3 Reconstruction

Recall that the reconstruction *must* be able to

1. operate in conjunction with the sampling techniques and
2. operate in real-time.

In addition, the reconstruction *should* should be

1. as accurate as possible;
2. as efficient as possible and
3. as fast as possible.

We have designed a reconstruction algorithm which is able to reconstruct the power spectral density of the sampled signal. We will reconsider the specifications one by one. First, the sampling techniques were carefully designed such that they adhere to this design. Therefore, the reconstruction algorithm is able to operate in conjunction with the sampling techniques. Second, the algorithm requires a specified amount of samples of the outputs of all cosets to operate. Since this amount can be specified, which is a trade-off between sampling time and accuracy, the reconstruction arguably operates in real-time. Third, Chapter 11 shows that for reasonable measurement time, the algorithm provides a reconstruction with NMSE less than $10^{-1.5}$. Although this may be accurate, the reconstruction does not provide a reconstruction as accurate as possible, since the complex interaction between the system modules implies that further optimisations of the algorithm may be possible. Finally, we have argued that the algorithm is computationally less intensive and less complex than other existing method based on optimisation or complex algorithms. We have optimised the operation of the reconstruction by exploiting sparsity and by providing a fast algorithm for generation of the matrices. By this means we tried to make the reconstruction as efficient and as fast as possible. However, this is also subject to discussion.

In addition to meeting the requirements, we have done extensive research on reformulating the algorithm in terms of using a biased estimate for the cross-correlations

of the output of the cosets. In comparison to the proposed unbiased estimator, a biased estimator can have smaller variance, which can result in smaller mean squared error of the estimation [19]. Although this analysis has been thorough, its conclusion was not definite. We therefore conclude to not include the results in this thesis. We consider research on using a biased estimator to be future work.

12.4 Detection

Recall that the detection of signals *must* be such that

1. detection of signals consists of determination of the set of frequencies which are occupied by signals other than noise;
2. the resolution of these frequencies can be specified and
3. the correctness of operation can be specified.

In addition, detection of signals *should* be such that

1. the resolution of detection can be as high as possible;
2. its operation can be as correct as possible and
3. its operation is as efficient as possible.

For the detection on the output of the reconstruction we have looked at two blind detection methods: modified energy detection and CAV. Based on a theoretical analysis we have decided to adopt the modified energy detector in our final system. This energy detector has been modified to work with our reformulated algorithm. This detector fulfills the must-haves: it is designed to detect per element of the reconstructed power spectral density whether noise or another signal is present. This also implies that it fulfills the should-have of the maximum possible resolution. From this set of comparisons, the set of frequencies occupied by some signal can be derived. Furthermore, it is possible to adjust the false-alarm probability of this detector and thus the correctness of operation can be specified.

From the results as presented in Section 11.4, we conclude that the detector does not fulfill the should-have that its detection should be as correct possible. The results indicate that relative small noise uncertainty considerably degrades the performance. Namely, the false alarm and the detection probability are affected by this problem. Furthermore, we have seen that this detector is not able to provide a reasonable[1] detection probability at low SNR's. Whether its operation is as efficient as possible is subject to discussion. From a computational point of view, the detector is efficient due to the fact that computation of its test statistic is relatively simple. From an operational point of view, however, there is room for improvement and other detectors may provide better performance in the sense of detection probability and false alarm probability. This point of view has also motivated the analysis of the CAV detector in Chapter 10. Due to time constraints the current

[1] A detection probability of 0.9 is given in the 802.22 EEE standard.

CAV implementation is limited in its detection abilities. That is, it cannot distinguish between frequency bands. Therefore CAV has not been included in our final system. Initial tests, however, indicate that the CAV detector outperforms the modified energy detector on the testing signal as described in Section 11.3. These tests motivate further research in the detection algorithm to be used in detection module.

Part II

Implementation

R.P. Hes
T.C. Leliveld
W.M. Melching

13 | Introduction

Part I provides an in-depth description of the theory required in the design of a high-performance spectrum sensing system. In this part, we will describe how this theory was implemented into a system that integrates sampling, reconstruction and detection and presents the results in a GUI in the form of a web-application to any user that connects.

In the following section we will state the system requirements that were set and have led to the software package in its final form. Then, starting a new chapter, the software platforms and libraries that were used will be briefly introduced, so that the reader might better understand their role in the system. We will also touch on the workflow that was followed during development. After this, we will move on the system itself and provide an overview of its architecture, so we can develop an understanding of its functioning in a structured manner. In the three chapters following this overview, we describe each of the system's three main individual components in greater detail. Next, the hardware that was used to obtain actual spectrum data for real-time analysis will be discussed. Consequently, we will perform an analysis of the quality and performance of the final software package, in which testing, formal guidelines and benchmarking will be deployed to determine how 'good' our software is. The part will be concluded with an overview of any future work that might be interesting.

All the code and documentation we have written for this project can be found on GitHub at `https://github.com/pd0wm/spectral`.

13.1 Specifications

This section gives a detailed description of the taken approach. This description consists of specifications according to which the approach will be designed. The specifications are divided into several categories. The categories are then analysed by MoSCoW prioritisation.

General

There are a number of general requirements that were considered at the start of the project:

- The implementation needs to be fast and efficient. Existing products are all targeting high performance. The advantage of compressive sensing is that less (expensive) hardware can be used at the cost of more demanding computations.
- The implementation needs to be modular in the sense that different kinds of approaches to the problem (both in hardware and algorithms) can be implemented and require a minimal adjustment in the software.
- The implementation needs to flexible enough that it could be run on different computational platforms (distributed networks, embedded devices, etc.) with minimal adjustment in the software.

To summarise, we target: *performance*, *modularity* and *flexibility*. More specific requirements, following the MoSCoW-method, like in Part I, are given in the following section.

Software

The software *must* be developed such that

1. it implements the generation and/or sampling of a simulated spectrum;
2. it implements the emulation of non-uniform sampling techniques on Nyquist-sampled spectra;
3. it implements the reconstruction of the sampled spectrum, using techniques from Chapter 8;
4. it implements the detection of any signal present in the reconstructed spectrum, using techniques from Chapter 10 and
5. it is able to clearly present the results of generation, reconstruction and detection to the user.

The software *should* be developed such that

1. the highest possible performance may be achieved;
2. it is designed in a structured and maintainable way and
3. it is be usable on multiple platforms.

Hardware

The software *must* allow for

1. the use of a Ettus Research USRP UHD N210 transceiver to obtain data;
2. the use of such a device to implement compressive sensing and

3. the use of two (or more) of these devices to implement collaborative sensing.

The software *should* allow for

1. fully utilising the USRP's capabilities.

For the *coulds* and *woulds* see Chapter 22.

14 | Preliminaries

14.1 Libraries

In this section we will discuss a number of used libraries and the advantages these have over other libraries/frameworks.

Python

Initially, two platforms were considered for development: MATLAB and Python. Taking our project preliminaries in account, Python provides the best option. Reasons for picking Python are:

- The Python libraries NumPy and SciPy use the same linear algebra libraries[1] that MATLAB uses for the actual calculations, so the speed difference will be marginal.
- Python is a non-proprietary platform, meaning it can be used commercially in the final product without additional charge.
- Python provides many libraries that allow for distributed networking/computing [28].
- Python is a full-blown programming language, meaning it has libraries for non-numerical applications such as web-frameworks, that allow for more advanced visualisation options.

The Python version used is 2.7. We are forced to use Python 2 (instead of the newer Python 3) because GNU Radio does not support Python 3. The original reference and used reference are respectively [29][30].

GNU Radio

GNU Radio is a comprehensive library of signal processing tools. It offers a framework and a vast collection of digital signal processing functions. The initial version of our system was written in this framework. By building custom signal processing

[1] Depending on OS this can be one of the many implementations. A few popular are openBLAS, ATLAS, Accelerate on Apple machines or Intel's proprietary MKL.

blocks we could extend the functionality of GNU Radio with our own compressive sensing functions.

However, soon this became too slow and complex for our needs. In the final version of our system we only used the USRP block from the GNU Radio library in one of our sources to receive samples from the USRP.

Numpy

NumPy is Python's standard numerical calculations library, providing a faster (multi-dimensional) array and high level bindings for a variety of linear algebra routines, amongst other. NumPy is licensed under the new-BSD[2] license. The reference manual is available at [31].

Scipy

SciPy is Python's replacement for a lot of default functions that are included in MATLAB. Relying on NumPy data structures, it gives a variety of subpackages (such as signal, their DSP-library) providing another means of quickly implementing advanced calculations. SciPy is licensed under the new-BSD[2] license. The reference manual is available at [31].

Flask

Flask is a micro web framework. The choice for a web interface will further be discussed in Chapter 18. The advantage of Flask over other web frameworks (such as Django) is that it is a more bare-bones minimalistic (i.e. simple) approach that suits our needs. Flask is licensed under the new-BSD[2] license. The reference manual is available at [33].

Twisted

Twisted is Python's event-driven network programming framework. It was used in conjunction with Autobahn. Twisted is licensed under the MIT[3] license. The reference is available at [34].

Autobahn

Autobahn is a real-time framework for websockets. It was used as a means of sending data to the GUI. It is written on top of Twisted and asyncio and is licensed under the MIT[3] license. The reference is available at [35].

[2]The BSD 3-Clause license is free for commercial and non-commercial use for more info and the template see [32].

Pyro

Pyro is a popular remote-object library. It allows us to do application programming over the network. In the future it could be an easy means of doing distributed networking but as of now it functions as a simple means of inter-process communication. Pyro is licensed under the MIT[3] license. The reference manual is available at [37].

Bootstrap

Bootstrap is a front-end CSS[4] and JavaScript framework. The framework was developed by Twitter. It implements a number of primitives, much like a GUI framework does on X11 (the display server used in many Linux distributions), allowing us to quickly implement a portable GUI. It suits our demands to be able to render both on mobile and desktop views, without much effort. Bootstrap is licensed under the MIT[3] license. The reference manual is available at [38].

Highcharts

Highcharts is a JavaScript plotting library capable of fulfilling all front-end plotting needs. This library was picked because of its customisability and ease-of-use. One major side note is that it is only free for non-commercial use. The Highcharts reference is available at [39].

14.2 Workflow

Version Control

For version control git was used. Git is a distributed version control system enabling non-linear development in the form of branching. This allows us to maintain a stable master branch, which is used to maintain a stable version of our product, a development branch in which features can be combined and minor bugs can be fixed and several feature branches for larger feature implementations. This specific workflow with git is called Git Flow [40]. We used a GitHub private repository to host our project on.

Testing

To verify the correctness of our code we used two kinds of testing: unit tests and integration tests.

[3] Software under the MIT license is free for both commercial and non-commercial use. For more info see [36].

[4] Cascading Style Sheets, a language for styling web pages.

Unit tests

Unit tests are small tests that test on a function level. They test if each function returns the right values for given input signals and tests for edge cases. These unit tests should be fast and be run often while writing code. This way, the correctness of one's code may be verified at all times. When the tests pass, the code is guaranteed to work according to the specifications. This becomes particulary important when working on a large project with multiple people. Any change in code can be verified quickly. Writing unit tests also forces the developer to think more about their code and stimulates modular design.

It is also possible to write the tests before the implementation of a function. This is called test driven development (TDD). In our design we decided not to use this, because the specifications were not entirely clear at the start of the project. In case of an abundance of additional time, TDD would definitely be a preferred approach.

Integration tests

Integration tests are tests that encompass a large part of the system. In our case we wrote tests where a couple of functions were tested at the same time. The output data was then compared by the output of a MATLAB script that we knew was correct.

Nose

In addition to the standard Python testing library Unittest, the library nose was used. This library provides some additional test commands and asserts which allow us to set up a more extensive test framework.

Profiling

For profiling the standard cProfile library was used. This is included in the standard Python distribution and is the de facto standard for profiling in Python.

15 | System Overview

In this chapter we will give an overview of the chosen system architecture. In the following chapters we will describe each part of the system in more detail. From the start we implemented a library (also called a package in the Python community) in which we build up our primitives. This was our first step in tackling the problem in a structured manner.

15.1 Model-View-Presenter

For the design of our library we chose a Model-View-Presenter architectural pattern (with passive view, referred to as MVP from now on). It is a variation on the well known Model-View-Controller pattern [41]. It consists of the following components:

Model This encompasses the sub-package spectral.core, that handles all the domain logic[1].

Presenter This encompasses the sub-package spectral.supervisor, that handles all the application logic[2].

View This encompasses the sub-package spectral.web, that contains the GUI elements.

An overview can be seen in Figure 15.1. The most important advantage of MVP is that the domain logic and application logic are completely separate. The model concerns itself with the domain logic, the view with the representation of the data and the presenter manages the data and user input synchronisation. This allows us to do orthogonal design.

Model

The goal was to implement enough primitives in these libraries that a large number of configurations can quickly be constructed together. This is to account for the

[1] The domain logic comprises the modelling of our problem.

[2] The application logic is the logic that drives the application. This is amongst other things handling the input data from the user and reflecting it in the model.

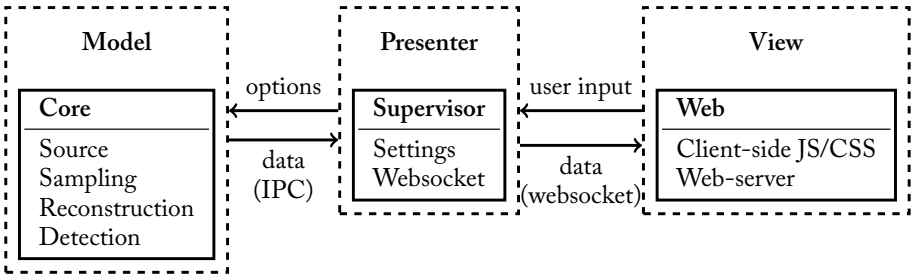


Figure 15.1: Illustration of the MVP pattern as an hierarchical way of separating our system.

variety of configurations in which this product can be used. It also allows us to test different kind of sampling techniques with reconstruction methods.

A side effect of writing code this way is that it makes our code modular. This means most "blocks" can be tested individually. To accommodate this, the Strategy design pattern is used for all the models. The Strategy pattern implements a family of algorithms behind an interface. It was first introduced in [42]. This is further discussed in Chapter 16.

View

The view should supply the user with a clear overview of what is happening in the model and allow them to change execution parameters during runtime. It fulfills the role of a *graphical user interface* or GUI. An additional requirement is that the view should be accessible to multiple users at the same time. How this is achieved will be described in Chapter 18.

Presenter

Finally, the presenter assumes the role of middle-man, in the sense that it forms the 'glue' between model and view. It coordinates both their actions and allows them to interact in a controlled manner. A major feature of the presenter is that it coordinates the entire system in a concurrent manner, in stead of sequentially, which should greatly increase performance, since most tasks can be handled simultaneously. The presenter will be described in-depth in Chapter 17.

16 | Model - The Core

In this chapter we will discuss the model part of our system. This consists of the various implementations of source, sampling, reconstruction and detection components. All functionality discussed in this chapter is contained in the spectral.core sub-package. Figure 16.1 shows the model and its interfaces. Every interface sets up a contract for one or more methods, which mean any class that implements this interface must also implement the method.

16.1 Source

All sources implement the interface `Source`, which has a contract for a generate method. This generate function should return the requested number of samples from a specific source. The sources can be divided in three categories, the simulated sources used for testing our algorithms, the file sources which generate data from saved files and real-world sources that connect to the USRPs to retrieve samples.

Simulated Source

The simulated sources are used for testing our algorithms. All the simulated sources have the same base class, `SimulatedSource`. This base class adds some utility methods to the sources to add Gaussian noise to the generated data.

Sinusoidal

The sinusoidal source generates a real valued signal, with a sum of real valued sinusoidal signals with different frequencies. Because the signal is real valued this generates a symmetrical spectrum. Optionally an SNR can be specified.

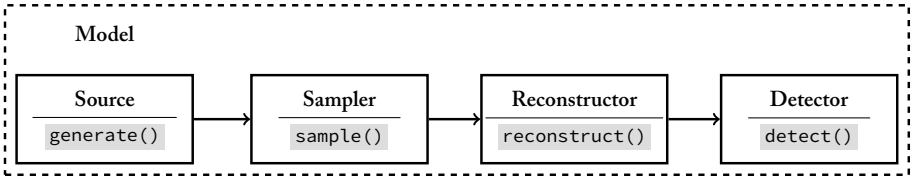


Figure 16.1: The model and its interfaces

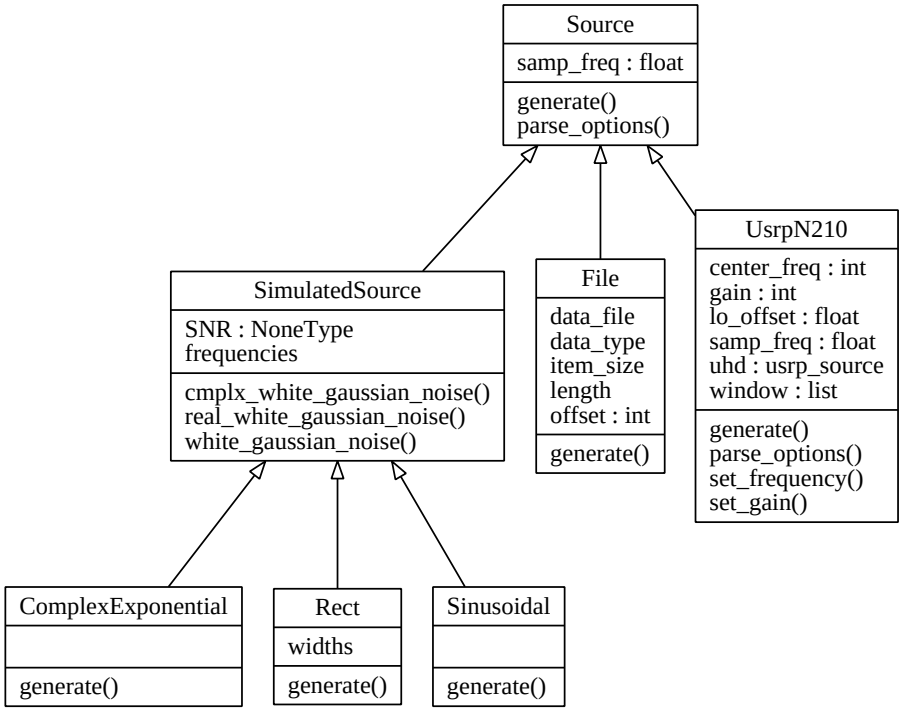


Figure 16.2: The UML diagram of the sources

Complex exponential

The complex exponential source is the complex equivalent of the sinusoidal source. The generated complex signal is the sum of complex exponentials with different frequencies. Optionally an SNR can be specified.

Rectangular source

The rectangular source generates a signal with multiple rectangles in the spectrum with specified frequencies and widths. The time domain signal is generated by adding multiple sinc functions with a carrier frequency. Optionally an SNR can be specified.

File source

The file source reads the samples from a file. This was used to test with real world data, but still get reproducible results. The files are compatible with GNU Radio's file sink. The file source loops over the entire data.

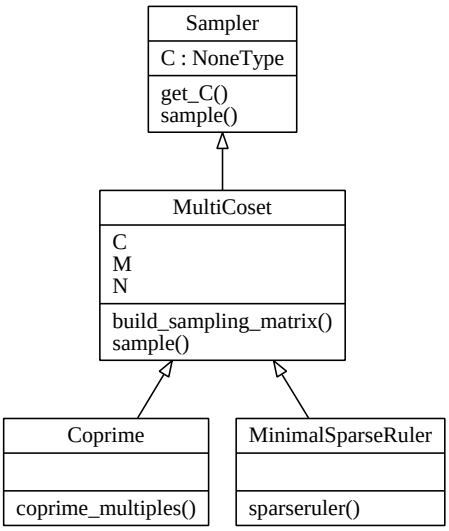


Figure 16.3: The UML diagram of the samplers

USRP N210 source

The USRP source uses the `finite_acquisition` function from the GnuRadio package. This provides a very easy way to get a specified number of samples from one USRP. This method has a few drawbacks (see Section 19.1). The problems with the DC compensation of the local oscillator are prevented by shifting the local oscillator out of the sampled spectrum. Unfortunately this halves our effective bandwidth. A function is also implemented which allows us to change the center frequency and the gain.

16.2 Sampling

After the uniform samples are generated, different sampling methods have to be applied. Each sampler implements an interface called `Sampler`, which has a contract for sample-method. The interface also defines a getter for the C-matrix[1].

Multi-coset sampler

Multi-coset sampling (as introduced in Chapters 7 and 9) is based on a multiple device sampler. The crucial component here is the offset of the common sampling period (sometimes referred to as the 'ruler' of a multi-coset system in context of the minimal sparse ruler problem). The list of offsets is one of the constructor arguments used to generate the C-matrix which has dimensions defined by M and N, which are the other arguments of this implementation.

[1]This is required for the reconstructors as their algorithms require knowledge about the sample intervals.

The implementation of the sample method takes a signal, uniformly sampled, as its argument and returns a multi-coset sampled signal. Our initial implementation used a loop and a matrix multiplication. To speed up this function the for loop was vectorised by a single matrix multiplication. This is achieved by reshaping the input signal to a matrix with one dimension N and the other determined by the length of the input signal.

Minimal sparse ruler sampler

The minimal sparse ruler inherits from the multi-coset sampler. It feeds a solution to the minimal sparse ruler to the multi-coset sampler, which it obtains from a lookup table. It inherits all the other functionality from the multi-coset sampler.

Coprime sampler

The coprime sampler also inherits from the multi-coset sampler. It generates a number of coprime multiples (see Chapter 9) as a set of intervals and feeds that into the multi-coset sampler. It inherits all the other functions from the multi-coset sampler.

16.3 Reconstruction

Two different kind of reconstruction methods are implemented. One is similar to the one specified in [2] and the other is discussed in Part I of this thesis. Both reconstructors implement the interface `Reconstructor`, which has a contract for a reconstruct method, which is depicted by the UML-diagram in Figure 16.4.

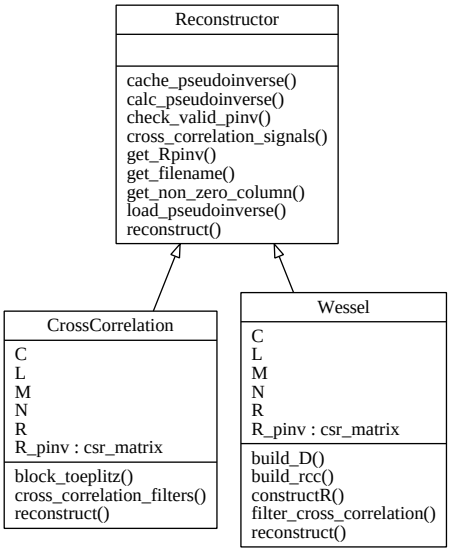


Figure 16.4: The UML diagram of the reconstructors

Both classes make use of shared functionality to calculate a pseudo-inverse. This is a wrapper around SciPy's `pinv`. Because this operation is quite time-consuming for larger matrices, a caching mechanism was introduced to lower start-up times.

CrossCorrelation

This class implements an algorithm based on the one described in [2]. The algorithm has two parameters. The first one determines the maximum lag that will be estimated of cross-correlation of the cosets, often referred to as L. The second one is the sampling matrix used by the sampler described in Section 16.2. These are also the parameters specified[2] in the paper.

Optionally a switch can be passed to disable the caching mechanism described in the base class. During the initialisation of the class it builds the R-matrix as discussed in [2] and computes its pseudo-inverse. This was implemented for debugging and testing purposes.

The reconstruct method is implemented by calculating the cross correlations of the non-uniformly sampled input signal. A helper function is defined in the `Reconstructor` interface that calculates the cross correlations of the input. To speed up this function the multiple correlations are vectorised into a number of matrix multiplications. The pseudo-inverse is then multiplied by these cross-correlations to reconstruct the signal.

Wessel

The reconstructor, as reformulated by Wessel Bruinsma, is a variation on the reconstruction technique implemented by `CrossCorrelation`. The theory is thoroughly described in Part I of this thesis. The algorithm takes the same parameters[2], namely the maximum lag estimated of the cross-correlation of the cosets and the sampling matrix used by the non-uniform sampler. This is required for the reconstruction algorithm.

Similar to the other reconstructor implementation, a switch to disable the caching mechanism has been implemented. This was also implemented for debugging and testing purposes. Like the other reconstructor, this reconstructor generates an R-matrix and computes its pseudo-inverse. Details on this algorithm may be found in Chapter 8.

The reconstruct method is implemented by calculating the cross correlations (again) of the non-uniformly sampled input and multiplying it with the pseudo-inverse.

From an implementation perspective, the structure of the pseudo-inverse of the Wessel algorithm maps better on the data structures of NumPy. NumPy's basic array data structures are in row-major order[3]. The final phase of both reconstruction steps is reshaping the cross-correlation matrix to a single column vector and multiplying

[2] There are actually four parameters: L, N, M and C. However, N and M can be determined from the shape of C.

[3] This is the standard way of saving an array in C, two notable exceptions that save their arrays in column major order are MATLAB and Fortran. Column major order is often called Fortran style.

it with the pseudo-inverse. This operation is more memory efficient in the Wessel reconstructor.

16.4 Detection

A single kind of detector is implemented. The Strategy pattern is still used because of possible future additions. All detectors have to implement the interface `Detector`, which has a contract for a detect method.

Arianandа

The Arianandа detector (named after the writer of [2]) implements the theory described in Chapter 10. It takes L, K, C, $\mathbf{R}^{\dagger}$, $\mathbf{R}_c$, window length and P_{fa} as its parameters. Its current implementation requires its input data to be originating from the `Wessel` reconstructor class[4].

It implements the detect method by calculating a threshold (referred to as γ_k in the theory) and comparing that with the PSD of the signal. It makes use of information of the wessel reconstructor and the sampler in question to make a better estimate of a threshold for the signal.

[4] It can also be implemented for `CrossCorrelation` but requires some adjustments. This has not been implemented.

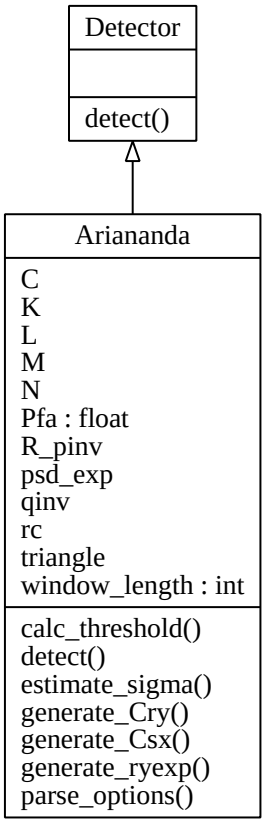


Figure 16.5: The UML diagram of the detectors

17 | Presenter - The Supervisor

In the Model-View-Presenter architectural pattern, the presenter forms the glue between the model and the view. The code is contained in the spectral.supervisor sub-package.

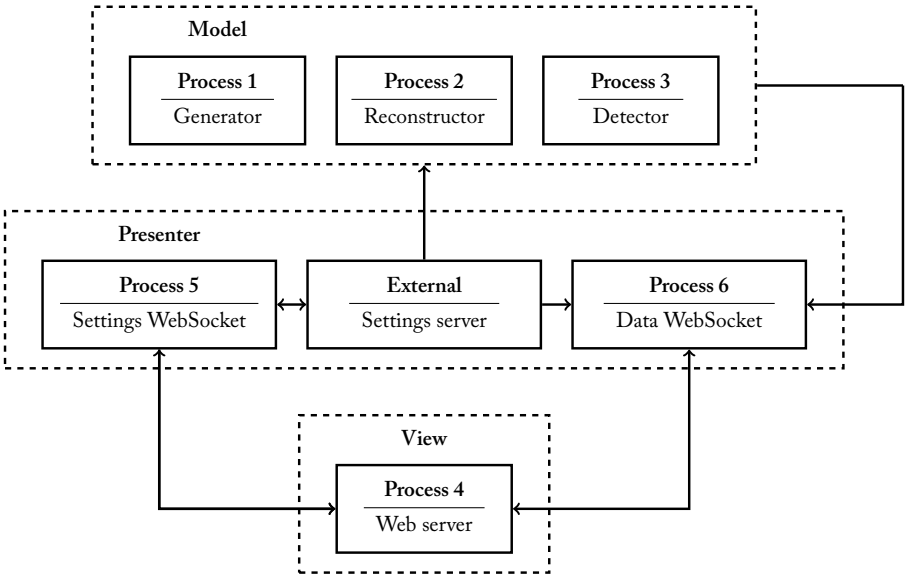


Figure 17.1: The presenter and the central role it fulfills in the system

The model notifies the presenter whenever its state changes, which will then trigger the presenter to update the view using the new data. At the same time, the view notifies the presenter whenever a user interacts with it, upon which the presenter will take the necessary actions to reflect this user-event in the model. These interactions are shown in Figure 17.1 and will be described in detail in the rest of this chapter.

The presenter's tasks can be summarized as follows:

- Coordinate model and view actions.
- Manage system-wide settings during runtime.
- Update the view using current model data.

Control of model and view will be described in Section 17.1. After this, the subject of system-wide execution parameters will be dealt with in Section 17.2. Finally, we will get into the matter of model-view interaction regarding settings and data in Section 17.3.

17.1 Primary control

Coordinating both model and view is primarily handled by the Python script run.py. This script may be executed with a number of command-line arguments:

f_samp Specifies the sampling frequency the USRPs should operate at.

L Specifies the L-parameter throughout the system, which determines the dimensions of various vectors and matrices.

source Specifies what module should be used as a data source for the system, choices are the USRP, various theoretical sources such as a sinusoidal and a locally saved dump file that was previously recorded.

snr When a theoretical source is selected, this specifies the amount of Gaussian noise that should be added to the signal. Used to test the system's performance.

dump When a local data dump is selected as source, this specifies where the dump file is located.

ip When a USRP is selected as source, this specifies the IP-address of the USRP that should be used for data-acquisition.

controlport Specifies the port at which the WebSocket (see Section 17.3) that handles control of the system should connect.

dataport Specifies the port at which the WebSocket that handles the data streams for external access should connect.

Using the parameters set by the user or a set of predefined defaults, The presenter will initialise each subcomponent in a separate process (using Python's `Process`, from the multiprocessing library in [43]), allowing tasks to be handled concurrently. The current implementation runs six processes alongside each other, whose behaviour is defined in processes.py Python script. We defined the following six processes:

1. The generator and sampler, this can be any combination of one of the sources in Section 16.1 and one of the samplers in Section 16.2.
2. The reconstructor, from Section 16.3.
3. The detector, from Section 16.4.
4. The web server, an important component of the view, as will be described in Chapter 18.

5. The control WebSocket server, which exposes the model's settings so that they can be modified from external applications (like the view).
6. The data WebSocket server, which exposes the model's currently processed data for external applications.

These processes need to be able to exchange data. This process is called inter-process communication (IPC). For example, the reconstructor process needs data generated by the generation process to perform its reconstruction. This is implemented using a number of Python `Queue`s (from Python's multiprocessing library in [43]), wrapped in a class that implements safe queuing and de-queuing by means of locks and `None` return types[1].

17.2 Settings server

Managing system-wide settings and allowing the user to modify them at runtime are also tasks of the presenter. System settings are made available between different Python processes using the Python package Pyro. This package enables us to use Remote Procedure Calls, which means we can use external objects over sockets as if they were local. In our system it functions mainly as a means of IPC on the local machine, but in theory it could be used as a central settings server from which a distributed network of computational machines get their settings and a number of views can push their settings to. This way of having system settings managed by a single object maps to the Observer design pattern [44].

17.3 WebSockets

To expose model settings and result data to external applications the system makes use of the WebSocket-protocol. The WebSocket protocol defines a full-duplex single socket connection that can be used to send messages between a client and server. The WebSocket specification was initially implemented in JavaScript, but nowadays interfaces are available in many other programming languages as well, including Python. Multiple packages even exist, of which Autobahn was chosen, because of its ease of use and clear examples.

Factories and Protocols

Autobahn implements the WebSocket specification using another widely used Python package called Twisted, which is an event-driven network engine. When using Autobahn, a WebSocket server is defined by a factory object and a protocol object. A factory is, simply put, an object that can create other objects, which is a basic object-oriented programming (OOP) design pattern [44]. A factory is created for both the WebSocket servers once, at runtime, when their respective processes start.

[1] The default queue raises an exceptions when trying to dequeue an empty queue, which is not desirable behaviour in this application.

From that point, either factory will create a protocol object for every client that connects to the server. The factory will perform some initialisation on the protocol and pass a reference of itself along with it, so every protocol instance shares the same factory object, more or less like a parent.

The protocols themselves define the behaviour of the WebSocket server upon various events. Both protocols handle the same events, which are relatively straightforward:

`onConnect` Triggered when a client tries to connect to the server.

`onOpen` Triggered after a connection has been successfully established.

`onMessage` Triggered when the server receives a message from the client.

`onClose` Triggered when a connection with a client is closed.

Settings

The settings WebSocket server uses Pyro to access the model's settings and makes them available to any client that connects at request. The settings WebSocket server is defined by the `ServerProtocolControl` protocol class and the `ServerProtocolControlFactory` factory class. Whenever a client connects (consecutively triggering `onConnect` and `onOpen`), the protocol will register itself with the parent factory. It will also send the connected client a message containing every current setting, so that the client may initialise itself correctly. Whenever a client sends an updated setting over the WebSocket connection (triggering `onMessage`), the protocol forwards the update via the Pyro settings object and makes sure any other client is notified of the changed setting via the parent factory, which relays the message to all clients that are currently registered by the factory. Finally, when a client disconnects, its associated protocol will unregister itself with the factory before going out of scope.

Data

The data WebSocket server publishes model data to any client that connects to it. The data WebSocket server is defined by the `ServerProtocolData` protocol class and the `ServerProtocolDataFactory` factory class. It functions in a similar manner as the settings WebSocket server, but maintains a direct data connection with the model using queues as described in Section 17.1, instead of using Pyro. The generator, reconstructor and detector processes each have their own queue to which they push the currently processed data, so that it can be made available on the WebSocket. The queues and buffers are both held by the parent factory, so every protocol will read the same data. The data WebSocket server does take any action when a client connects. Registering itself with the parent factory is not needed because connected clients do not interact with each other. Any connected client may send the server a request for a certain data stream, coming from the generator, reconstructor or detector process. When this happens, the associated protocol will

ask the parent factory to update its data buffer for the correct process. Finally, the container will be encoded into a string using JSON[2] and send to the client.

[2]JavaScript Object Notation, a standardised way to express objects using a string.

18 | View - Visualisation & Control

The last part of the system design consists of a web-based graphical user interface (GUI) that allows for easy control and visual verification of the system and its status. This component therefore fulfills the role of view in the design's Model-View-Presenter architecture. It is contained in the sub-package spectral.web.

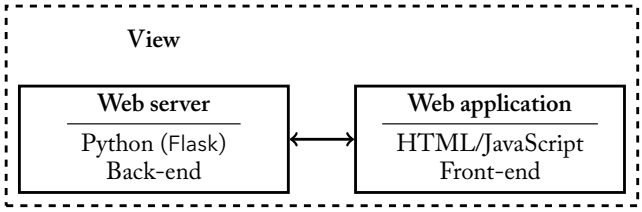


Figure 18.1: The view

The view can be roughly divided into a server-side part (or *back-end*) and a client-side part (or *front-end*). This is shown by Figure 18.1. The server-side system, written in Python, prepares the user interface for display on the client-side. The client-side system is a web application, written in JavaScript, that handles the actual presentation of the data, as well as any action the end-user might perform which should have an effect back in the functional core.

The major advantage of using a GUI in the form of a web application is its (almost) guaranteed cross-platform support, as well as relatively easy development and debugging, since most browsers are shipped with more than adequate development tools. The choice for Python to write the back-end in was made so that the Flask Python package could be used as web server. Flask allows the programmer to quickly set up a simple web-server to serve the web application with. The front-end itself is developed in JavaScript, which is the de-facto scripting language for client-side web-development.

This chapter describes each of the individual parts of the visualisation software. First, the web-server (back-end) is dealt with, an indispensable component that makes the web application accessible within the network for whoever wants to connect. This section will describe how the web-interface is constructed by the server before being presented to the user and will touch on subjects like the use of templating and predefined elements to generate a web-page. Then, the switch to client-side is made. This section will uncover the inner workings of the front-end as it is presented to the user, in terms of both visualisation and control.

18.1 Back-end

The back-end's tasks can be summarized briefly as follows:

- Handle client connections.
- Serve the web application to any client that connects.

Both tasks are largely handled by the Python package Flask, which was introduced in Section 14.1. Using Flask, a series of routes[1] may be defined. Clients can then send a request to one or more of these routes, for example to request a web page like the visualisation front-end. Flask will execute the code that is associated with the specified route upon receiving a request, like rendering the visualisation front-end, after which the result of this execution is sent back to the client.

Templating

For rendering a web page, like the front-end, Flask makes use of yet another Python package called Jinja2. This package is a so-called *templating engine*, which renders a web page using a predefined layout, containing some static content, but also an arbitrary number of expressions that are interpreted and executed by Jinja2 upon receiving a request to generate dynamic content.

18.2 Elements

Jinja2 can be used to generate an entire web page at once, but also to render a single element of the page, such as a slider or a plot. In the design of the visualisation package, it was deemed preferable to easily swap in and out specific elements, thereby obtaining a fully modular system. The web application is therefore set up as a skeleton containing for example the page header and footer, as well as a grid-like body that is used to hold the chosen page elements. This way the front-end is completely modular. The programmer only has to pick a certain element, configure it to display and/or modify the correct data or setting and choose its location in the grid. The templating engine will render each element individually and then insert them into the page body. The elements that were implemented and can be used to construct the web application in this matter are listed below.

Text element Being the most simple element available, it is capable of displaying a short string value. Such a string might reflect, for example, the status of a certain component in the model. The text element is not a control element and therefore its data is only directed from server to client.

[1] A route defines a request on a specific URL.

Figure 18.2: Text Element

Checkbox element The checkbox element allows the user to modify a simple boolean value in the model's settings. As an example, it might be used to toggle a certain functionality. Being a control element, it maintains a bi-directional connection with the presenter, so it can notify the presenter whenever a user modifies its value and receive updates whenever another client modifies its associated value. View-presenter interaction will be covered in Section 18.3.

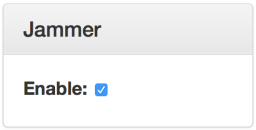


Figure 18.3: Checkbox Element

Slider element A more flexible element that allows to user to set and observe the value of a certain (numerical) parameter in the system. Like the checkbox, this is a control element that is bi-directionally connected to the presenter.

Figure 18.4: Slider Element

Visualisation element The last and most complex one available, this element presents the current results of any of the three sub-processes as carried out by the functional core (generation/sampling, reconstruction, detection). The user can pick any of the three datasets to display, which may then be displayed using a real-time spectrogram or an FFT-style plot, or, when detection data is selected, a column diagram that shows at what frequencies an actual signal is determined to be present. The spectrogram is developed by our team and allows the user to observe how the frequency spectrum varies over time by depicting a certain frequency content with a certain colour shade. The FFT-plots and detection column graphs are generated by the JavaScript plotting library Highcharts. The code for both plots mainly consists of setting up Highcharts for the correct display of the incoming data, preparing the incoming data for display (apply scaling, etc.) and calculating the correct plot limits to yield a clear picture. The FFT-plot is also equipped with averaging functionality, which can be used to smooth the rather variable input data for a more stable image.

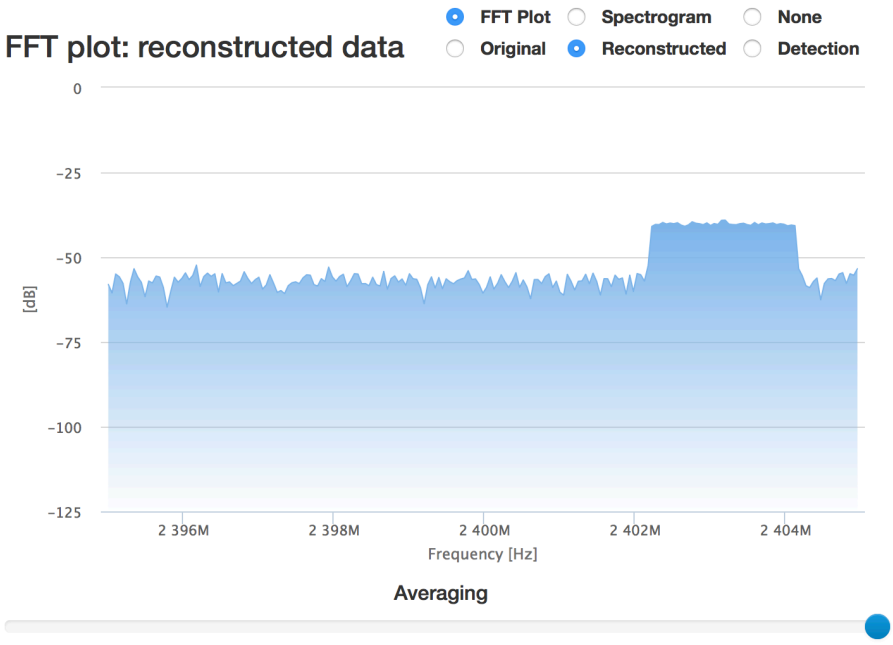


Figure 18.5: FFT Visualisation Element

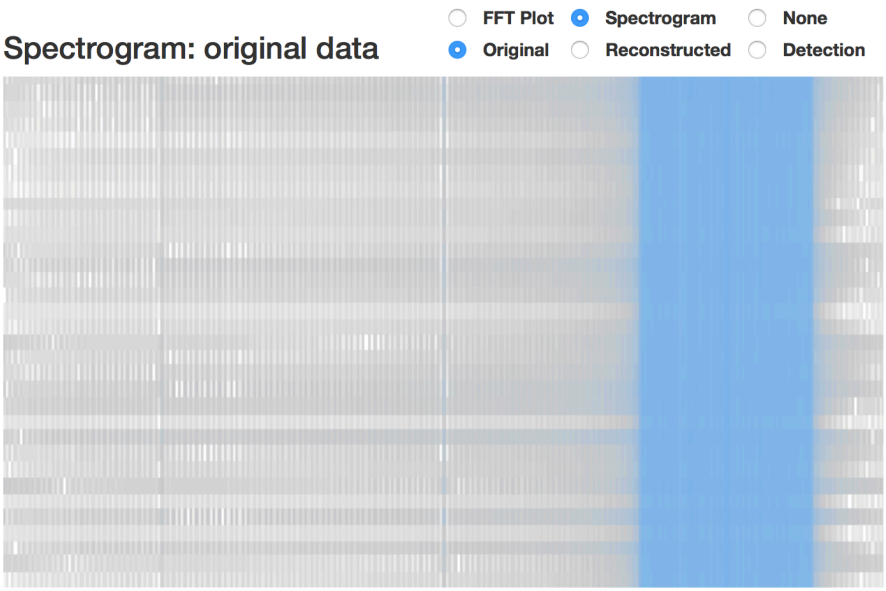


Figure 18.6: Spectrogram Visualisation Element

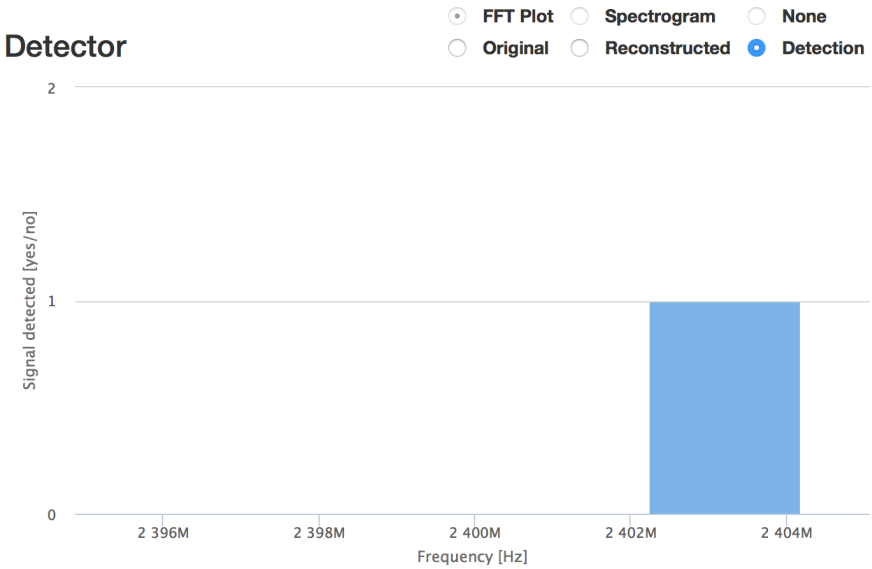


Figure 18.7: Detection Visualisation Element

Each of the elements consists of server-side and client-side logic, which is written in Python and JavaScript, respectively. The server-side logic consists of a class that contains some of the element's properties, like a unique ID, title and a reference to a Jinja2 template as previously described, that is rendered whenever the element is used (when the client requests the web application). The client-side logic is a JavaScript object that defines the element's dynamical behaviour, such as handling user-input and/or server-sent updates.

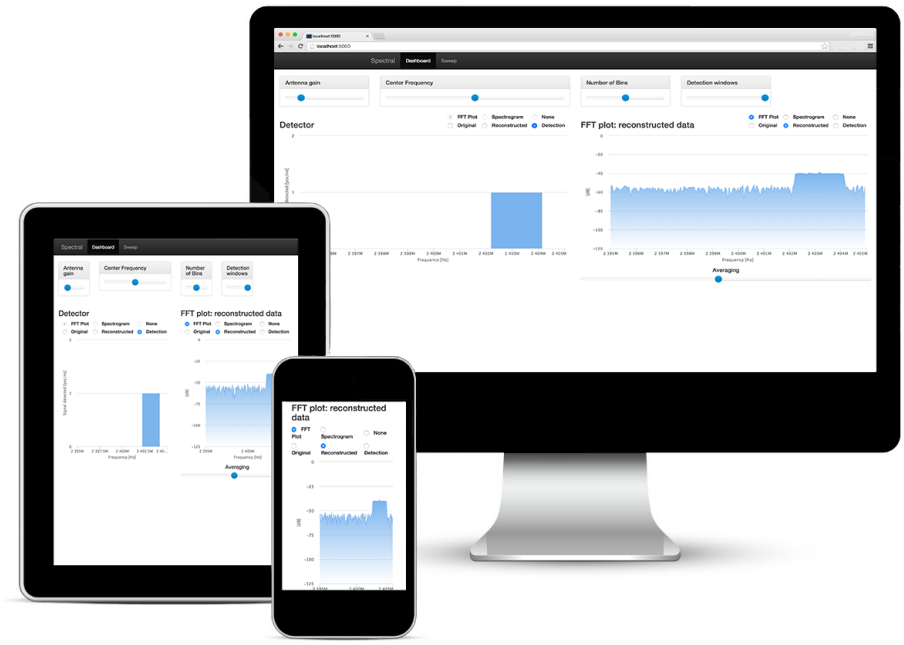


Figure 18.8: The graphical user interface on multiple devices

Figure 18.8 shows a sample configuration of the view using the described elements on multiple devices.

18.3 Front-end

As mentioned, the front-end consists of a web page containing the elements as generated by the server. The user is presented with a certain combination of the elements that were described in the previous section, that allows them to manipulate settings back in the model and verify its results. The front-end is largely based on Bootstrap, which provides a solid base to build the rest of the interface on. Bootstrap provides the HTML (HyperText Markup Language, used to define the structure a web page), that defines the grid in which all elements are consequently rendered. Bootstrap also handles the scaling of the entire application for different screen sizes, so the application remains usable, even on smartphones.

Presenter interaction

The model has to be updated upon any user interaction that takes place in the view. At the same time, UI-elements in the view should be able to access the most recent data from the model. For this to happen, a connection has to made between model and view. In Chapter 17 we introduced the settings and data WebSocket servers, which were built for this goal.

The web application connects to both of these WebSockets using JavaScript's built-in WebSocket implementation, wrapped in the `Control` and

`Visualisation` objects, respectively. These two objects are implemented following the *Singleton*-pattern, which is a design-pattern that ensures that only a single instance of the given object may exist at all times (bearing similarities to a `static` instance of a class in some other languages), which is useful since both `Control` and `Visualisation` handle all WebSocket related actions for the entire web application [42, p. 127].

The `Control` object connects to the settings WebSocket server from Section 17.3. Whenever a user modifies a setting, e.g. by altering a slider value, `Control` will send a message over the WebSocket to the presenter to notify it of the update. After this, the presenter will publish the update to the model. The various subcomponents of the model may implement a method to read the most recent system settings that might concern them to process setting updates that might concern the. The presenter will also broadcast the to all other connected clients. Upon receiving such an update, the `Control` object will execute the update routine of the appropriate element, for example moving the slider to the newly received value. This process ensures that the current program settings are synchronised among every client.

The `Visualisation` object connects to the data WebSocket server from Section 17.3. It keeps track of the currently shown data by registering each plot present in the application. Whenever a plot is registered, `Visualisation` will send a message over the WebSocket, requesting the presenter to send the current data of the required type. The presenter will respond by sending an object containing the requested data and some additional meta-data. Upon receiving this data, `Visualisation` will pass it through to the element that first triggered the request, which will then re-render the plot. After this, `Visualisation` will, again, send a request to the presenter, restarting the cycle. This process will continue until the user picks another data type for display, or disables the plot altogether, which will trigger `Visualisation` to unregister the plot and safely destroy it. Carefully keeping track of every plot in the application and the data type it displays like this prevents memory-leaks and bandwidth wasted on unprocessed plot data.

19 | Hardware

19.1 USRP N210

Introduction

The hardware sampler used in our system is a USRP N210, a software defined radio (SDR) produced by *Ettus Research*. This radio is equipped with the SBX front-end, which has frequency range of 400 MHz to 4400 MHz that can both transmit and receive simultaneously. This signal that enters the radio is first amplified and then mixed down by a two local oscillators with a 90 phase shift. Then the resulting two signals are sampled by two 100 MS/s ADCs resulting in an I and a Q signal. Because the local oscillators have a 90 phase shift the resulting signal is quadrature or IQ sampled. With the information from both signals we can measure the complex envelope of the signal.

The USRPs can also be used as a transmitter using the DAC and the same local oscillators. We used this during the testing of our design to generate test signals. However this is not used in our final design.

These radios are connected to a PC by using gigabit Ethernet, thus limiting the effective sample rate to 25 MS/s with a 16 bit resolution or 50 MS/s with an 8 bit resolution. If you want to use the full potential of the internal ADC (2x 100 MS/s) and DAC (2x 400 MS/s) you have to write your own firmware for the internal FPGA.

Drivers

The communication between the USRPs and our software is handled by the UHD driver[1] written by *Ettus Research*.

In the first version of our software we built everything using GNU Radio. We used the gnuradio-uhd blocks to get samples from our USRPs. Soon we switched to a pure Python version, but we still used the gnuradio-uhd library to retrieve our samples. We used a specific helper function `gnuradio.uhd.finite_acquisition` from the GNU Radio libraries. This helper function started a low level stream, waits for n samples, closes the stream and returns the samples to the caller.

[1] The code for this driver can be found online at: `https://github.com/EttusResearch/uhd`.

The USRPs have an automatic calibration for the DC offset introduced by the local oscillator after the mixer. However due to a bug in the USRPs, every time a stream is started this DC calibration is reset. The effects of this bug can be resolved by limiting the bandwidth and shifting the local oscillator frequency out of the observed spectrum. This effectively halves our usable bandwidth.

To work around this bug we had to write our own low level code to stream samples from the device. We wrote C++ code to connect to the USRPs, synchronise them and start a stream. The received samples are then send through a socket to the rest of the system for further processing. This was not an easy task, but has the added benefit that our system no longer depends on GNU Radio and we have very low level control over the precise operation of the USRPs.

20 | Software quality & Testing

20.1 Introduction

In this chapter we will discuss the methods we used to write a large software product. Traditionally this is not part of a bachelor Electrical Engineering. However we feel the importance of proper code quality is one that every engineer should harness in the ever more software oriented industry. To achieve this we consulted some classic works on the subject. Examples of this are

- Code Complete 2: A practical handbook of software construction [45];
- Clean Code: A handbook of agile software craftsmanship [46];
- The Art of Unix Programming [47];
- Learning Python [48];
- Dive into Python [49];
- Design Patterns: Elements of Reusable Object-oriented Software [42].

Firstly the importance of testing immediately became clear. The second observation is the importance of doing good design upfront. The choice of correct data structures and algorithms makes all the difference. This can be seen in the previous chapters when we refer to design patterns. A third theme is maintainability. This is something that lives through proper documentation and code quality and will be discussed below as well. Lastly we wanted to evaluate advice from Code Complete 2, a standard work on programming techniques, and how this applies to the current state of our software stack.

20.2 Testing

Unit testing is a way of doing verification on code by checking each function with set of known inputs and outputs. Integration tests are a means of checking a set of modules/functions working together for known inputs and outputs.

With a combination of unit testing and integration testing, the code can be verified on the function level. With the help of special tools you can specify the expectations about the working of a function, in our case the Python unittest and nose library.

Table 20.1: A table showing the coverage of our tests

Name	Statements	Miss	Cover
spectral.core	5	0	100 %
spectral.core.detection	2	0	100 %
spectral.core.detection.arianandа	72	5	93 %
spectral.core.detection.detector	5	2	60 %
spectral.core.helperfunctions	38	2	95 %
spectral.core.reconstruction	2	0	100 %
spectral.core.reconstruction.crosscorr	42	5	88 %
spectral.core.reconstruction.reconstructor	54	26	52 %
spectral.core.reconstruction.wessel	46	1	98 %
spectral.core.sampling	4	0	100 %
spectral.core.sampling.coprime	17	0	100 %
spectral.core.sampling.minimalsparse	11	0	100 %
spectral.core.sampling.multicoset	21	0	100 %
spectral.core.sampling.sampling	7	0	100 %
spectral.core.source	5	0	100 %
spectral.core.source.complex_exponential	10	0	100 %
spectral.core.source.filesource	21	0	100 %
spectral.core.source.rect	16	0	100 %
spectral.core.source.simulatedsource	24	11	54 %
spectral.core.source.sinusoidal	10	0	100 %
spectral.core.source.source	7	2	71 %
spectral.core.source.usrp_n210	42	32	24 %
TOTAL	461	86	81 %

The functions are then evaluated with predefined input out put pairs, such that their behaviour can be verified.

For the verification we also used coverage. This extension on our unit testing library also keeps track of which lines of code are run during your tests. If certain lines of code are never run, you know your tests are incomplete. An example of this can be found in Figure 20.1. In this output a certain branch of an if statement is never taken.

Sometimes it is very easy to write specifications for the correct output of a function. However with certain functions in the reconstructor or the detector, the only way to verify the results was to check them according to known output matrix. These matrices were generated by a MATLAB script that we assumed was correct[1].

Our current tests cover 81 % of our model code. Most of the untested parts involve IO operations (caching of the matrices, filesources and USRP's) but there are still some functional tests missing, most notably the ones for `CrossCorrelation`. The results can be seen in Table 20.1.

[1]These MATLAB scripts were written by the people who were working on the theory.

```
52          if (ruler_length - 1) not in sparseruler:
53              raise NotImplementedError("Values other than " + str(sparseruler.keys()) + " not implemented.")
54          return sparseruler[ruler_length - 1]
```

Figure 20.1: Coverage output of a branch that is never taken

20.3 Documentation

For the documentation we used Sphinx. Sphinx is a documentation generator that can build a API reference from the docstrings in our code[2]. Sphinx makes use of Python's introspection property which allows it to get all the dosctrings and generate an automated API reference from this. This reference is supplied alongside this report.

20.4 Qualities of good code

We will discuss the quality of our code according to the definitions from Code Complete 2 about external and internal characteristics of quality code. External qualities are qualities the user of the software will notice. Internal qualities are only noticed by the developers of the software.[45]

External qualities

"*Correctness* is the degree to which a system is free from faults in its specification, design, and implementation." For our design we used extensive testing to verify the correctness of our code and its results. Our algorithms are base on peer reviewed articles from established authors. This is further confirmed by our automated testing.

"*Usability* is the ease with which users can learn and use a system." To easy the adaptability of our system we provide an extensive API documentation of every function of our code. This makes it easy to use the existing functions to expand the system to your own needs.

"*Efficiency* is the minimal use of system resources, including memory and execution time." Our design is built with the goal of high performance (see Chapter 21). While writing the code we thoroughly optimised our program by making use of profiling. We identified bottlenecks and removed them if possible. We also tried to optimise our algorithms and data structures (e.g. using sparse matrices).

"*Reliability* is the ability of a system to perform its required functions under stated conditions whenever required — having a long mean time between failures." From the Art of Unix programming[47]: "The rule of repair: when you must fail, fail noisily and as soon as possible". In our code we have multiple checks on input data, when the data is not correct it notifies the user and shuts down. With this method we found bugs immediately, and this resulted in very stable system.

[2] A docstring is a comment on the first line of a function indicated with the triple quote.

"*Integrity* is the degree to which a system prevents unauthorised or improper access to its programs and data. The idea of integrity includes restricting unauthorised user accesses as well ensuring that data is accessed properly — that is, that tables with parallel data are modified in parallel, that date fields contain only valid dated, and so on." When transferring data between different processes we use thread-safe queues, or remote objects. This ensures that no race conditions or other issues with parallel data access can happen. We did not focus on preventing unauthorised access of our data when building this system, this is something that has to be considered when someone else then the user has access to the system.

"*Adaptability* is the extent to which a system can be used, without modification, in applications or environments other than those for which it was specifically designed." Because we have written our system in python our code is runnable on all hardware that is supported by Python. This includes all personal computers and even embedded devices with a recent ARM or MIPS processor.

"*Accuracy* is the degree to which a system, as built, is free from error, especially with respect to quantitative outputs. Accuracy differs from correctness; it is a determination of how well a system does the job it's built for rather than whether it was built correctly." Before building our system we wrote a specifications for our system. Using testing we can verify that our system adheres to these specifications.

"*Robustness* is the degree to which a system continues to function in the presence of invalid inputs or stressful environmental conditions." Our system is not really designed with robustness in mind. However there are multiple fail safes in the system. For example when the sampler does not return the requested number of samples, the frame is discarded and a new one is requested.

Internal qualities

"*Maintainability* is the ease with which you modify a software system to change or add capabilities, improve performance, or correct defect." Our system is designed with modularity in mind. All components live in their own module and are not dependent on each other. This makes it very easy to change all small part of the system without interfering with the other parts. By making use of the strategy pattern, different solutions for one problem (e.g. sampling) can be easily added.

"*Flexibility* is the extent to which you can modify a system for uses or environments other than those for which it was specifically designed." The system we built is usable for all kinds of high performance signal processing. Instead of samplers, reconstructors and detectors other kind of signal processing modules can be implemented. Also by using the Model-View-Presenter pattern the code is loosely coupled. For example, it is very easy to write another GUI for the system while keeping the other logic intact.

"*Portability* is the ease with which you can modify a system to operate in an environment different from that for which it was specifically designed." See *Adaptability*.

"*Reusability* is the extent to which you and the ease with which you can use parts of a system in other systems." For all our signal processing blocks we wrote documentation and an API manual. This makes it easy for others to use our modules in their own system.

"*Readability* is the ease with which you can read and understand the source code of a system, especially at the detailed-statement level." Our code is written in Python, which is designed to produce very readable code. When our code is not immediately clear to the reader comments are added to improve legibility. Special documentation strings are added to each function to explain the working of each function and explain its arguments.

"*Testability* is the degree to which you can unit-test and system-test a system; the degree to which you can verify that the system meets its requirements." Our functional core is almost completely under test. Only the code that interfaces with physical hardware is not testable. The functionality of the reconstruction and the detector is checked with the output of the MATLAB scripts written by theory group.

"*Understandability* is the ease with which you can comprehend a system at both the system-organisational and detailed-statement levels. Understandability has to do with the coherence of the system at a more general level than readability does." By using proven design patterns (Model-View-Presenter) to design our system architecture our system becomes more comprehensible. By separating parts of the system it becomes much easier to reason about a piece of code and its working.

21 | Performance

In this chapter the performance of the implementation will be discussed. First we will introduce the measurements performed to evaluate said performance after which we will discuss some practical implementations of these optimisations.

21.1 Profiling

A good metric for performance are profiling reports. Profiling reports describe the time spent inside functions (execution time). They allow us to identify bottlenecks in the system which slow down calculations. More often than not these have allowed us to make crucial speed-ups in the system.

21.2 Algorithms and Data structures

Our initial step was to improve the algorithms and data structures involved. This is usually seen as the first best way of improving performance. We used profiling reports as an indicator to identify bottlenecks and improve the algorithms or data structures only there were necessary as to keep the complexity of our code low[1].

Sparse matrices

After the initial profiling reports it became clear that the dot product with the pseudo-inverse in the reconstructor (see Section 16.3) formed a bottleneck. After doing some inspection on the pseudo-inverse it appeared to be rather sparse. This sparsity allowed us to use SciPy's optimised sparse data structure. We chose the rather standard `csr_matrix` data type. As noted by [31], the advantages of this data structure are, amongst other, fast matrix vector products and better memory efficiency. The speed difference for the reconstruct method is shown in Figure 21.1.

[1] As Donald Knuth, considered as the father of informatics, once said: "We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified."[50]. The essence of this quote is that one should only start optimizing code once it has been proven to be necessary as this usually increases the complexity of the code and thus the maintainability. Maintainability is often only an afterthought but imminent for code that has to last.

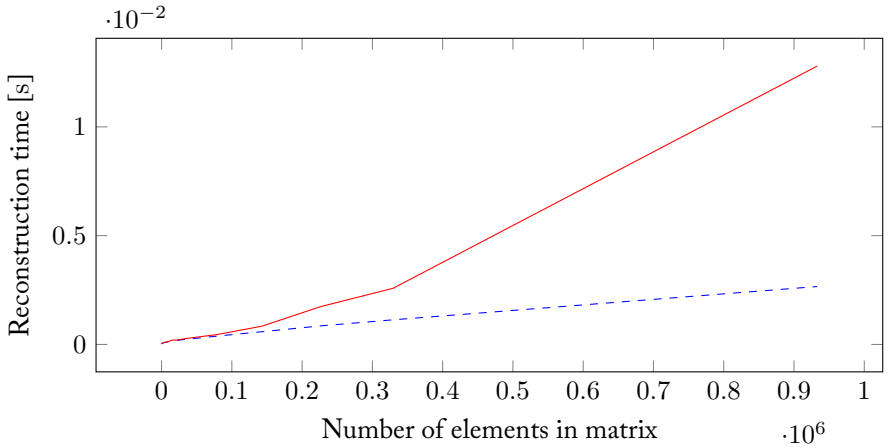


Figure 21.1: Reconstruction time (normalized over 1000 runs) versus number of elements in matrix of sparse (dashed) and standard matrix (solid) data structure

Vectorisation

Another bottleneck was formed by the calculation of signal cross-correlations. There are a number of ways to perform this calculation:

- Expressed as a number of matrix multiplications.
- Numpy's `correlate` function.
- SciPy's `fftconvolve` function.

Figure 21.2 shows that each technique has its own area of best performance. The matrix multiplication works best for vector lengths of about 50 elements. Numpy's `correlate` performs best from 50 until 400 elements and for everything above that `fftconvolve` is the fastest.

21.3 Multiprocessing

Each major part of the model (source, sampling, reconstruction and detection) can be run concurrently by means of implementing a pipeline. This is done by means of multiprocessing. We spawn a separate process for each major computation: the source & sampler, the reconstructor and the detector. This information is sent using inter-process communication (IPC). This introduces some overhead and delay, but has the major advantage of allowing us to make use of multiple CPU cores. Ultimately, it depends on the configuration of the parameters whether multiprocessing is advantageous, because IPC (which has to do memory transfers through L3-cache) can be the bottleneck on less demanding parameter sets. Results of a standard program run can be seen in Table 21.1. In this case a speed up of 48 %[2] is achieved by the use of multiprocessing over serial execution.

[2]This is largely going to be dependent on the platform it was run on. In this case a mid 2015 Macbook Pro with an i7-4870HQ was used to run this benchmark.

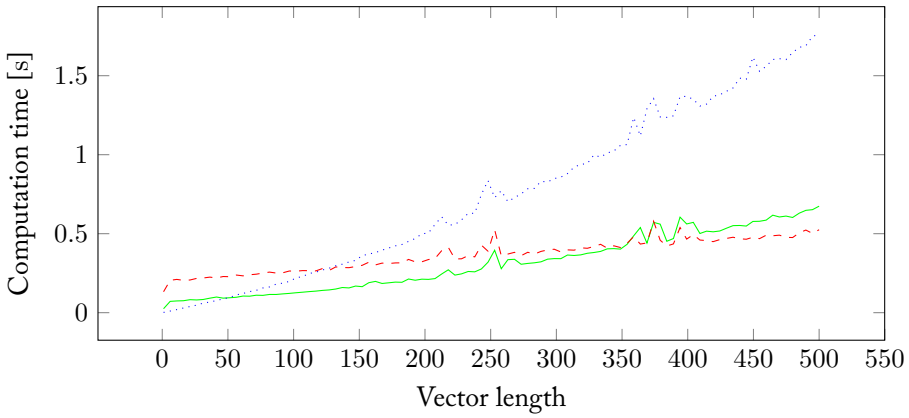


Figure 21.2: Computation time for 100 cross-correlations versus vector length. Solid is NumPy's `correlate`, dashed is SciPy's `fftconvolve` and dotted is the matrix implementation.

Table 21.1: Reconstruction times for $N = 51$, $L = 3$, $K = 2000$, with an OFDM file source, a minimal sparse ruler sampler and the Wessel reconstructor, normalized over 1000 runs

Method	Single reconstruction time
Serial	3.60 ms
Multiprocessing	2.43 ms

An issue that arose is that the current implementation of Apple's accelerate, is not compatible with multiprocessing. This is due to a bug[3] on Apple's end. Instead, OpenBLAS was used instead at a very marginal performance loss.

21.4 Results and Further Improvements

All in all, a system has been implemented that allows for real-time compressive spectrum sensing. Future work can be sought in the implementation of additional techniques for parallel processing. CUDA and OpenCL are good candidates for this (especially because of the sparse structures involved, which are ideal GPU applications). Furthermore, a good study of the memory structures throughout our calculations could prove to be beneficial for further speed-ups.

[3] As noted on the NumPy's issue tracker this is an issue Apple is not addressing. A workaround is to use a Python 3 feature which allows process spawning rather than forking which solved this issue. Unfortunately due to dependencies on GnuRadio on earlier stages of this project Python 2 was used. For more information see `https://github.com/numpy/numpy/issues/5752`.

22 | Future Work

As of now we have build a flexible software framework. We split the implementation in part hardware and part product implementations.

22.1 Hardware Implementations

There are a number of configurations discussed in Chapter 9 that can be applied with different advantages and disadvantages.

Circular-sparse-ruler hardware implementation

A hardware implementation of samplers combined in a setup discussed in Section 9.3. It allows for a high performance setup with optimal reconstruction resolution. This implementation is focused on high-performance compressive spectrum sensing.

Coprime hardware implementation

A hardware implementation of two samplers in coprime setup discussed in Chapter 9. It would involve two samplers syncing on a combined period whilst the individual sample frequencies of the samplers would have a coprime ratio. The use case would involve optimal performance for least cost. Coprime uses non optimal "ruler" but is implementable with two samplers only.

The software coprime truncater uses two input sample vectors, with a different number of samples from two samplers with different sample rates. The sampling matrix is constructed as with the normal coprime sampler. The two input vectors are then transformed into one output vector that matches the sampling matrix.

Synchronisation of USRPs to implement coprime sampling

For our implementation of co-prime sampling we needed to synchronise the sampling of two USRPs. This can be accomplished by connection them with a special MIMO cable. This allows them to internally synchronise their sampling clock. It even allows them to schedule commands at exactly the same time, so they can start sampling at exactly the same moment.

To verify that our synchronisation is correct, we dumped two frames from the same moment in time to a file. If we plot them on the same time scale, as in Figure 22.1, we can see that they align almost perfectly. If we zoom in on the second burst in the signal, see Figure 22.2, we can see the offset due to the local oscillator offset.

The only offset we have is from the phase offset between the two local oscillators of the USRPs. After each tune command the two USRPs have a random phase offset of -180 to $+180$. We don't know yet if that influences our results. If that is the case we can calculate this lag by looking at the cross-correlation between the two USRPs or by transmitting a reference pulse after each tune command and calculating the delay.

At this moment the synchronised sampling works, but something is going wrong in the processing or visualisation of the data. We did not have the time to find these problems and solve them. This is something for future work.

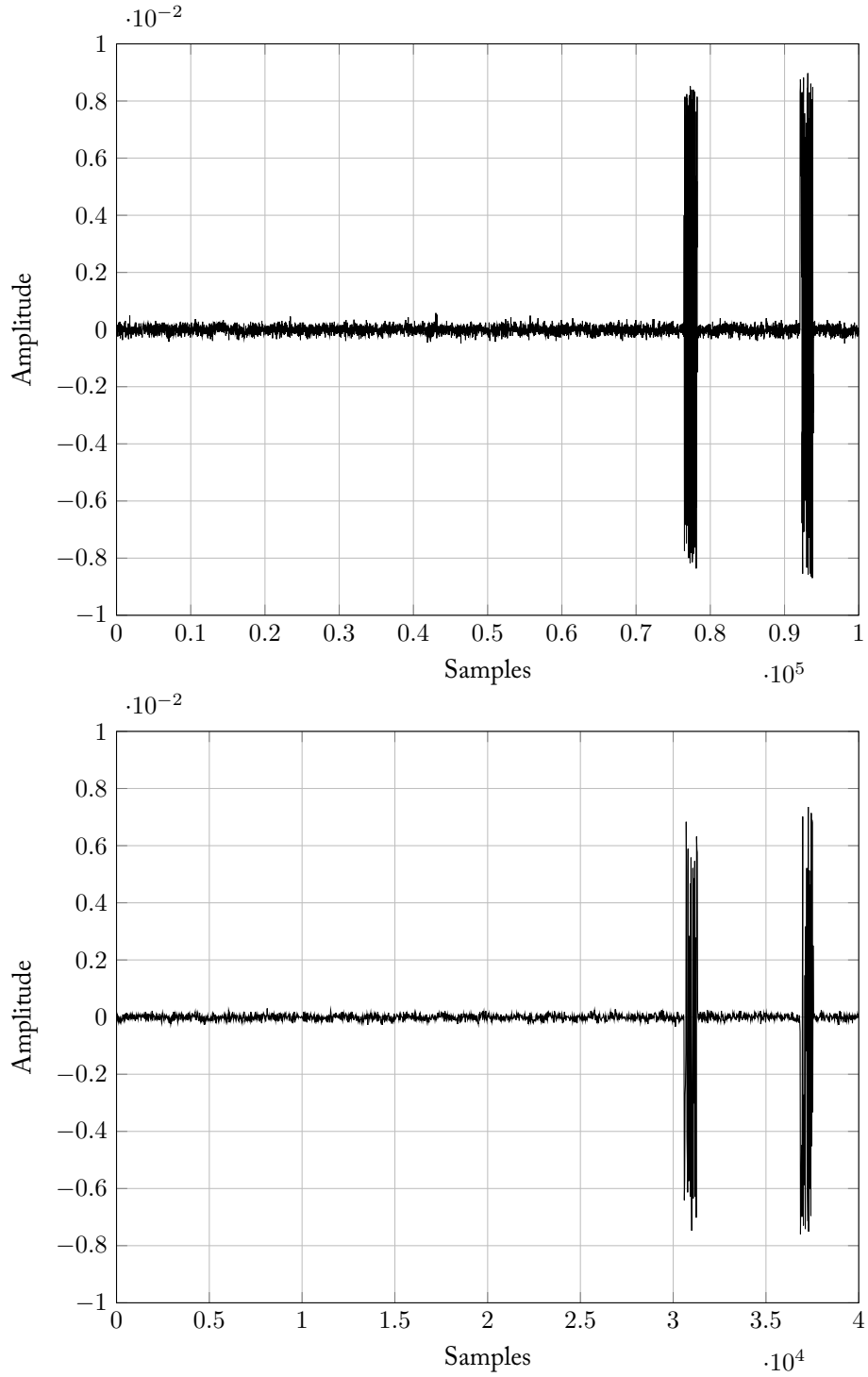


Figure 22.1: Output of the two samplers

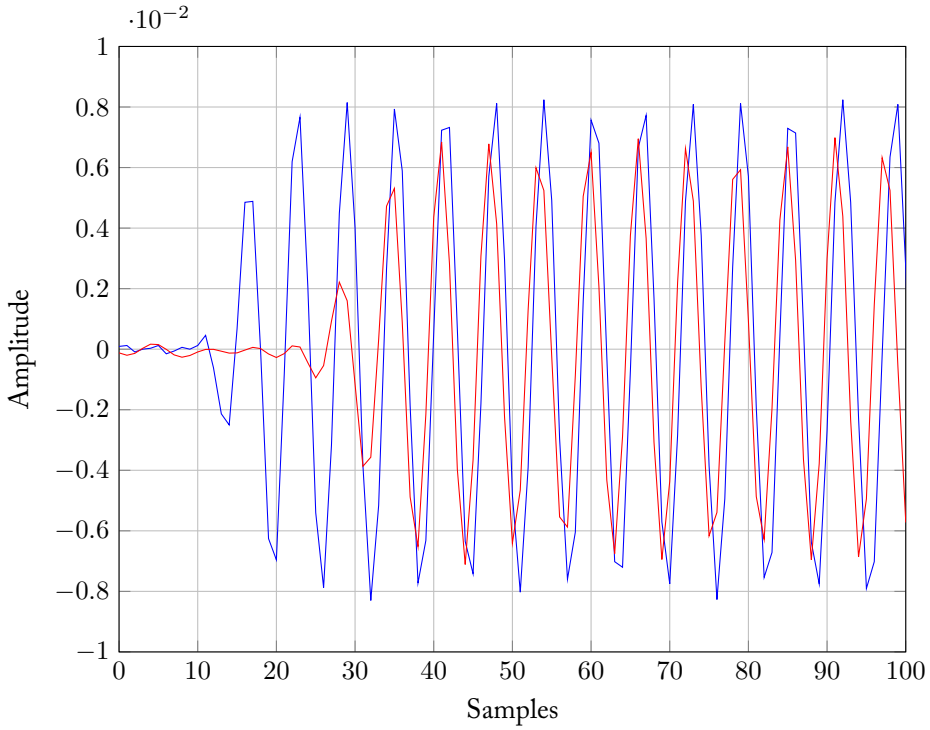


Figure 22.2: Zoomed in version of the start of the second burst

22.2 Product Implementations

Distributed network

In the current implementation a lot of multiprocessing is done as discussed in Chapter 17. A variation on this is to run a distributed network of nodes all reconstructing part of or the entire spectrum and recombining this on the views. This distributes the computing power amongst several nodes while the views have to do very little processing. Because there are several libraries available (such as Pyro4) which allow to do this kind of computing on a standard LAN network existing networks can be used for this purpose.

Mobile platform

Our system could also be used in a distributed system that uses mobile devices (e.g. mobile phones) for collaborative spectrum sensing. This enables mobile network providers to keep track of a database of the current spectrum usage. This could then be used for dynamic spectrum allocation where devices can communicate on the unused part of the licensed spectrum if the primary user (i.e. the user who paid for the license) is not using the specific frequency.

23 | Discussion

In this chapter we will reflect on the specifications given in Section 13.1.

General

Recall that the general requirements were:

- The implementation needs to be fast.
- The implementation needs to be modular.
- The implementation needs to flexible.

By using multiprocessing and profiling, we achieved a fast and efficient implementation of our algorithms. By using proven design patterns we obtained a modular and maintainable code base. Due to the use of Python, our solution is portable and flexible.

23.1 Software

Recall that the software *must* be developed such that

1. it implements the generation and/or sampling of a theoretical spectrum;
2. it implements the emulation of non-uniform sampling techniques on Nyquist-sampled spectra;
3. it implements the reconstruction of the sampled spectrum, using techniques from Chapter 8;
4. it implements the detection of any signal present in the reconstructed spectrum, using techniques from Chapter 10 and
5. it is able to clearly present the results of generation, reconstruction and detection to the user.

All of the above *musts* have been implemented. Therefore the requirements have been met.

Also recall that the software *should* be developed such that

1. the highest possible performance may be achieved;
2. it is designed in a structured and maintainable way and
3. it is be usable on multiple platforms.

Equivalent to the general requirements, these *shoulds* have been taken into account when designing and writing the software and their accompanying requirements have been met.

23.2 Hardware

Recall that with respect to the hardware the software *must* allow for

1. the use of a Ettus Research USRP UHD N210 transceiver to obtain data;
2. the use of such a device to implement compressive sensing and
3. the use of two (or more) of these devices to implement collaborative sensing.

Almost all of the above *shoulds* have been met. Only collaborative sensing has only partly been implemented. We have achieved the synchronisation of two USRPs, but were not able to implemented the reconstruction step in time (see Section 22.1).

And recall that the software *should* allow for

1. fully utilising the USRPs capabilities.

Our PCs are not able to compute the various steps in the algorithms fast enough to handle the $25\,\mathrm{MS/s}$ coming from the USRP. The computations could be sped up by finding more efficient algorithms or by using more powerful and/or distributed hardware for the computations.

Part III

Final Thoughts

24 | Conclusion

After approximately two months, we can put our minds to rest and conclude our final work. Despite the division of this work into two parts, we consider it important to reflect upon those parts in one conclusion, since together, they form our final product.

We have developed a theoretical framework whose specification are mostly met. Its performance has been evaluated and we can conclude that the framework provides the theoretical necessities to enable real-time spectrum sensing. Subsequently, we have developed a modular software toolkit which implements the developed theoretical framework. Together their performance has carefully been assessed. Courageously, we dare to say we have made a good attempt at answering the question whether we can develop a toolkit that enables real-time high-performance wideband spectrum sensing.

This thesis has been the desired opportunity demonstrate the knowledge we have accumulated the past three years. Although it may be a small step forward in the area of wideband spectrum sensing, we hope that we have encouraged the reader to follow the path we set out.

Bibliography

[1] D. Arianandа, G. Leus, and Z. Tian, "Multi-coset sampling for power spectrum blind sensing", in *Digital Signal Processing (DSP), 2011 17th International Conference on*, Jul. 2011, pp. 1–8.

[2] D. D. Arianandа and G. Leus, "Compressive wideband power spectrum estimation", *Signal Processing, IEEE Transactions on*, vol. 60, no. 9, pp. 4775–4789, 2012.

[3] E. Candes and J. Romberg, "Sparsity and incoherence in compressive sampling", *Inverse problems*, vol. 23, no. 3, p. 969, 2007.

[4] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information", *Information Theory, IEEE Transactions on*, vol. 52, no. 2, pp. 489–509, 2006.

[5] Y. Zhang, "On theory of compressive sensing via l1-minimization: Simple derivations and extensions", *TR08-11, CAAM, Rice University*, vol. 27, 2008.

[6] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling", *Signal Processing Magazine, IEEE*, vol. 25, no. 2, pp. 21–30, 2008.

[7] D. Dony Arianandа, D. Romero, and G. Leus, "Cooperative compressive power spectrum estimation", in *Sensor Array and Multichannel Signal Processing Workshop (SAM), 2014 IEEE 8th*, Jun. 2014, pp. 97–100. DOI: 10.1109/SAM.2014.6882349.

[8] P. Pal and P. P. Vaidyanathan, "Coprime sampling and the music algorithm", in *Digital Signal Processing Workshop and IEEE Signal Processing Education Workshop (DSP/SPE), 2011 IEEE*, IEEE, 2011, pp. 289–294.

[9] M. Lexa, M. Davies, and J. Thompson, "Multi-coset sampling and recovery of sparse multiband signals", Tech. Rep., Jan, Tech. Rep., 2011.

[10] L. Couch, *Digital & Analog Communication Systems: International Edition*. Pearson Education Limited, 2013, ISBN: 9780273774228. [Online]. Available: https://books.google.nl/books?id=4_WoBwAAQBAJ.

[11] S. M. Kay, *Fundamentals of statistical signal processing, volume 2: Detection theory, ser*, 1998.

[12] E. Axell, G. Leus, E. G. Larsson, and H. V. Poor, "Spectrum sensing for cognitive radio: State-of-the-art and recent advances", *Signal Processing Magazine, IEEE*, vol. 29, no. 3, pp. 101–116, 2012.

[13] X. Han, W. Xu, K. Niu, and Z. He, "A novel wavelet-based energy detection for compressive spectrum sensing", in *Vehicular Technology Conference (VTC Spring), 2013 IEEE 77th*, Jun. 2013, pp. 1–5. DOI: 10.1109/VTCSpring.2013.6691840.

[14] Y. Zeng and Y.-C. Liang, "Spectrum-sensing algorithms for cognitive radio based on statistical covariances", *Vehicular Technology, IEEE Transactions on*, vol. 58, no. 4, pp. 1804–1815, May 2009, ISSN: 0018-9545. DOI: 10.1109/TVT.2008.2005267.

[15] B. Bayar, N. Bouaynaya, and R. Shterenberg, "Kernel reconstruction: An exact greedy algorithm for compressive sensing",

[16] S. Kirolos, J. Laska, M. Wakin, M. Duarte, D. Baron, T. Ragheb, Y. Massoud, and R. Baraniuk, "Analog-to-information conversion via random demodulation", in *Design, Applications, Integration and Software, 2006 IEEE Dallas/CAS Workshop on*, IEEE, 2006, pp. 71–74.

[17] J. Li, Z. Wu, H. Feng, Q. Wang, and Y. Liu, "Greedy orthogonal matching pursuit algorithm for sparse signal recovery in compressive sensing", in *Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, 2014 IEEE International*, May 2014, pp. 1355–1358. DOI: 10.1109/I2MTC.2014.6860967.

[18] Y. L. Polo, Y. Wang, A. Pandharipande, and G. Leus, "Compressive wideband spectrum sensing", in *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, IEEE, 2009, pp. 2337–2340.

[19] M. Hayes, *Statistical digital signal processing and modeling*. John Wiley & Sons, 1996, ISBN: 9780471594314.

[20] Z. Quan, S. Cui, A. H. Sayed, and H. V. Poor, "Optimal multiband joint detection for spectrum sensing in cognitive radio networks", *Signal Processing, IEEE Transactions on*, vol. 57, no. 3, pp. 1128–1140, 2009.

[21] S. Kapoor, S. Rao, and G. Singh, "Opportunistic spectrum sensing by employing matched filter in cognitive radio network", in *Communication Systems and Network Technologies (CSNT), 2011 International Conference on*, Jun. 2011, pp. 580–583. DOI: 10.1109/CSNT.2011.124.

[22] A. Sahai, R Tandra, and M Mishra, "Spectrum sensing: Fundamental limits", *Draft of the book chapter in Cognitive Radios: System Design Perspective*, 2009.

[23] R. Pickholtz, D. Schilling, and L. Milstein, "Theory of spread-spectrum communications–a tutorial", *Communications, IEEE Transactions on*, vol. 30, no. 5, pp. 855–884, May 1982, ISSN: 0090-6778. DOI: 10.1109/TCOM.1982.1095533.

[24] J. Gevargiz, P. Das, and L. Milstein, "Adaptive narrow-band interference rejection in a ds spread-spectrum intercept receiver using transform domain signal processing techniques", *Communications, IEEE Transactions on*, vol. 37, no. 12, pp. 1359–1366, Dec. 1989, ISSN: 0090-6778. DOI: 10.1109/26.44207.

[25] R. Sharma and J. Wallace, "Correlation-based sensing for cognitive radio networks: Bounds and experimental assessment", *Sensors Journal, IEEE*, vol. 11, no. 3, pp. 657–666, Mar. 2011, ISSN: 1530-437X. DOI: 10.1109/JSEN.2010.2058097.

[26] S. Atapattu, C. Tellambura, and H. Jiang, *Energy Detection for Spectrum Sensing in Cognitive Radio*. Springer, 2014.

[27] Y.-C. Liang, Y. Zeng, E. Peh, and A. T. Hoang, "Sensing-throughput tradeoff for cognitive radio networks", *Wireless Communications, IEEE Transactions on*, vol. 7, no. 4, pp. 1326–1337, Apr. 2008, ISSN: 1536-1276. DOI: 10.1109/TWC.2008.060869.

[28] *Parallel processing and multiprocessing in python*. [Online]. Available: https://wiki.python.org/moin/ParallelProcessing.

[29] G. Rossum, "Python reference manual", Amsterdam, The Netherlands, The Netherlands, Tech. Rep., 1995.

[30] *Python reference manual*. [Online]. Available: https://docs.python.org/2/reference/.

[31] E. Jones, T. Oliphant, P. Peterson, *et al.*, *Scipy: Open source scientific tools for Python*, [Online; accessed 2015-06-20], 2001–. [Online]. Available: http://www.scipy.org/.

[32] *The bsd 3-clause license*. [Online]. Available: http://opensource.org/licenses/BSD-3-Clause.

[33] *Flask: Web development, one drop at a time*. [Online]. Available: http://flask.pocoo.org.

[34] *Twisted matrix labs: Building the engine of your internet*. [Online]. Available: https://twistedmatrix.com/trac/.

[35] *Autobahn python reference*. [Online]. Available: http://autobahn.ws/python/.

[36] *The mit license (mit)*. [Online]. Available: http://opensource.org/licenses/MIT.

[37] *Pyro: Python remote objects*. [Online]. Available: https://pythonhosted.org/Pyro4/#.

[38] *Bootstrap: Designed for everyone, everywhere*. [Online]. Available: http://getbootstrap.com.

[39] *Highcharts: Make your data come alive*. [Online]. Available: http://www.highcharts.com.

[40] V. Driessen, *A successful git branching model*, Jan. 2010. [Online]. Available: http://nvie.com/posts/a-successful-git-branching-model/.

[41] A. Syromiatnikov and D. Weyns, "A journey through the land of model-view-design patterns", in *Software Architecture (WICSA), 2014 IEEE/IFIP Conference on*, IEEE, 2014, pp. 21–30.

[42] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995, ISBN: 0-201-63361-2.

[43] Python Software Foundation, *The python standard library*. [Online]. Available: https://docs.python.org/2/library/index.html.

[44] Microsoft, *Observer*. [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649896.aspx.

[45] S. McConnell, *Code Complete*. Pearson Education, 2004.

[46] R. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Pearson Education, 2008, ISBN: 9780136083252. [Online]. Available: https://books.google.nl/books?id=_i6bDeoCQzsC.

[47] E. S. Raymond, *The art of Unix programming*. Addison-Wesley Professional, 2003.

[48] M. Lutz, *Learning Python*, ser. Safari Books Online. O'Reilly Media, 2013, ISBN: 9781449355692. [Online]. Available: https://books.google.nl/books?id=4pgQfXQvekcC.

[49] M. Pilgrim, *Dive into python*, 2000.

[50] D. E. Knuth, "Structured programming with go to statements", *ACM Comput. Surv.*, vol. 6, no. 4, pp. 261–301, Dec. 1974, ISSN: 0360-0300. DOI: 10.1145/356635.356640. [Online]. Available: http://doi.acm.org/10.1145/356635.356640.

[51] R. Yates and D. Goodman, *Probability and stochastic processes: A friendly introduction for electrical and computer engineers*. John Wiley & Sons, 2005, ISBN: 9780471272144. [Online]. Available: http://books.google.nl/books?id=eN4mAQAAIAAJ.

[52] R. G. Gallager, "Circularly-symmetric gaussian random vectors", 2008.

[53] L. Rugini, P. Banelli, and G. Leus, "Small sample size performance of the energy detector", *Communications Letters, IEEE*, vol. 17, no. 9, pp. 1814–1817, Sep. 2013, ISSN: 1089-7798. DOI: 10.1109/LCOMM.2013.080813.131399.

Part IV

Appendix

A | Ethical paragraph

This chapter is concerned with the ethical aspects of our business plan. The development of our product, a toolkit that enables high-performance wideband spectrum sensing, has several possible consequences. This ethical paragraph will first discuss some of those possible consequences, to continue with a description of our policy to address those issues, and other issues that have to be addressed in a business plan.

A first possible consequence of introducing our technique is a rapid acceleration of developments in Cognitive Radio which, eventually, may lead to a wide-spread use of Cognitive Radio networks. This in turn, may lead to increased interference in networks of licensed users. Those users then experience deteriorated network performance, which may hurt their applications. Another possible consequence of our technique is related to its misuse. Since the technique allows for fast wideband spectrum sensing, it also allows for fast tracking of user signals. This tracking could be used to effectively eavesdrop communication on a large scale.

We take an utilitarian point of view at the responsibility of application of this product. Since we can not exactly foresee all possible applications, nor do we have the capability to influence this, we will only take responsibility for the product itself. We expect that telecom agencies will regulate wide-spread use of wideband spectrum sensing and will therefore represent the interests of, for example, telecom providers, civilians and the military. By adhering to the local legal requirements of telecommunications we expect that misuse of our toolkit is reduced to a bare minimum, while maintaining a commercially viable situation for our company.

Environmental and safety issues are of less concern in our business plan, since our company is targeted at developing and maintaining a software toolkit. It does not focus on the actual hardware on which it may be used, and it therefore does not need a production line which entails those problems. Furthermore, our company can be regarded as a high tech company, and thus our employees will have to possess highly specialised knowledge in the area of wideband spectrum sensing and high-performance software development. We will therefore not impose gender quota, since this can conflict with the need to hire highly educated employees and may have severe consequences for the operation of the company.

B | Derivation of the reconstruction algorithm

In this appendix we derive the algorithm discussed in Chapter 8. The derivation aims to be an accessible, yet complete explanation which discusses all details necessary for implementation. Table B.1 presents an overview of the variables. Some variables are not discussed yet and the meaning of their names will become clear during the derivation of the algorithm.

Variable	Description
M	Number of cosets
N	Downsampling factor such that every coset samples the input signal once every N samples
L	Support of $r_{y_i,y_j}[m]$. This means that $r_{y_i,y_j}[m] = 0$ for $\|m\| > L$.
$x[n]$	Input signal
$r_x[m,n]$	Autocorrelation of the input signal
$y_i[n]$	Output of coset i
$r_{y_i,y_j}[m,n]$	Cross-correlation of the outputs of cosets i and j
$c_i[n]$	Signal which determines the output of coset i
$c_{i,j}[m]$	Deterministic cross-correlation of $c_i[n]$ and $c_j[n]$
$z_i[n]$	Helper variable related to $x[n]$ and $y_i[n]$
$r_{z_i,z_j}[m,n]$	Cross-correlation of $z_i[n]$ and $z_j[n]$
$\mathbf{r}_x[k]$	Aggregation of $r_x[(k+1)N-1]$ to $r_x[kN+1]$
$\mathbf{c}^-_{i,j}$	Aggregation of $c_{i,j}[-N+1]$ to $c_{i,j}[-1]$
$\mathbf{c}^+_{i,j}$	Aggregation of $c_{i,j}[1]$ to $c_{i,j}[N-1]$
$\mathbf{r}_{y_i,y_j}$	Aggregation of $r_{y_i,y_j}[L]$ to $r_{y_i,y_j}[-L]$
$\mathbf{r}_x$	Aggregation of $r_x[LN]$ to $r_x[-LN]$
$\mathbf{R}_{i,j}$	Convolution matrix of $c_{i,j}[m]$
$\mathbf{r}_y$	Aggregation of $\mathbf{r}_{y_M,y_M}$ to $\mathbf{r}_{y_0,y_0}$
$\mathbf{R}$	Aggregation of $\mathbf{R}_{M,M}$ to $\mathbf{R}_{1,1}$
n_i	Variable such that $c_i[n] = 1$ for $n = n_i$ and zero otherwise

Table B.1: Overview of the variables used

Consider the input signal to be a random stochastic process. To relate the output of a coset to the input signal, we need some mathematical constructs.

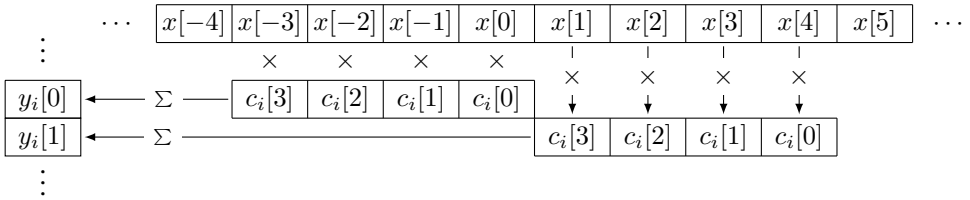


Figure B.1: Relationship between $y_i[n]$, $x[n]$ and $c_i[n]$ in the case that $N = 4$

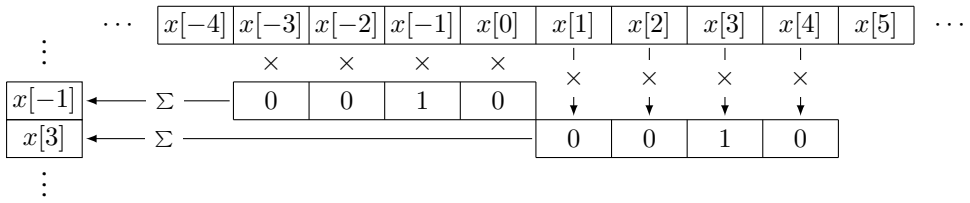


Figure B.2: Relationship between $y_i[n]$, $x[n]$ and $c_i[n]$ in the case that $N = 4$ and $c_i[n] = 1$ for $n = 1$ and otherwise zero

B.1 Construction of the output of a coset

In Section 8.3 we have seen that every coset samples the input signal every N samples. However, different cosets sample at different moments. To keep the way a coset samples the input signal as general as possible, we associate every coset i with a signal $c_i[n]$ which specifies the way coset i samples the input signal. This signal $c_i[n]$ is constructed such that $c_i[n] = 0$ for $n < 0$ and $n > N - 1$. To illustrate the way $y_i[n]$, $x[n]$ and $c_i[n]$ are related, suppose that $N = 4$. In this case, Figure B.1 shows how $y_i[n]$ is related to $x[n]$ by $c_i[n]$. That is, every group of N samples of the input signal yields one sample of the output of a coset. This is done by multiplying the samples in that group sample-wise by $c_i[N - 1]$ to $c_i[0]$ and then summing them. Mathematically,

$$y_i[n] = \sum_{k=(n-1)N+1}^{nN} x[k]c_i[nN - k]. \tag{B.1}$$

Equation (B.1) precisely relates the output of coset i to the input signal.

Consider the case that $c_i[n] = 1$ for a single n and otherwise zero. Then $y_i[n]$ consists of the sample of every group of N samples of $x[n]$. This means $y_i[n]$ is an N-decimation of the input signal. Figure B.2 illustrates this concept in the case that $N = 4$ and $c_i[n] = 1$ for $n = 1$ and otherwise zero.

To analyse Equation (B.1), we need a helper variable. Let

$$z_i[n] = (c_i * x)[n] \tag{B.2}$$

where $*$ denotes the convolution operator. The definition of the convolution oper-

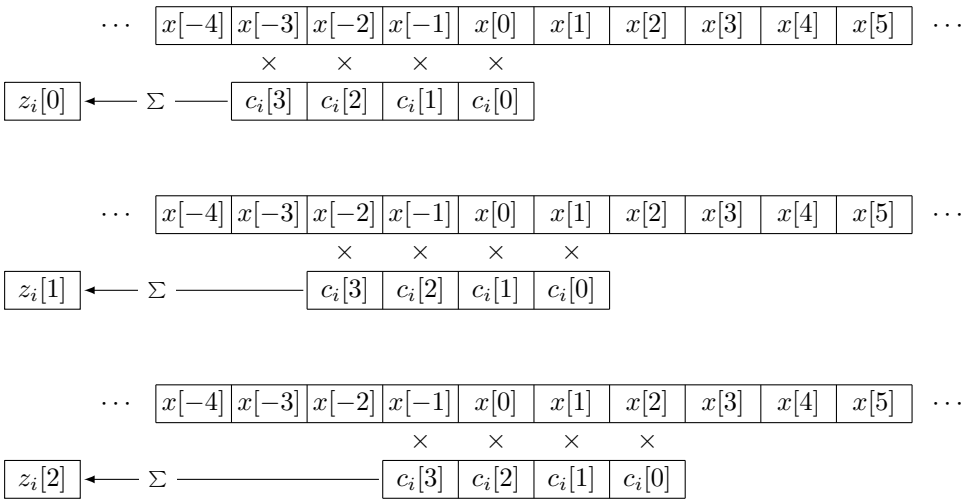


Figure B.3: Relationship between $z_i[n]$, $x[n]$ and $c_i[n]$ in the case that $N = 4$

ator yields that

$$z_i[n] = \sum_{k=\infty}^{\infty} x[k]c_i[n-k] \tag{B.3}$$

$$= \sum_{k=n-N+1}^{n} x[k]c_i[n-k], \tag{B.4}$$

since $c_i[n-k] = 0$ for $k < n-N+1$ and $k > n$. Figure B.3 shows how $z_i[n]$, $x[n]$ and $c_i[n]$ are related in the case that $N = 4$. By inspection of Equation (B.1) and Equation (B.3), we obtain that

$$y_i[n] = z_i[nN].$$

B.2 Autocorrelation and cross-correlation

The goal of the reconstruction method is to reconstruct the autocorrelation of the input signal given the output of all cosets. The autocorrelation of the input signal is given by

$$r_x[n,m] = E(x^*[n]x[n+m]).$$

We assume that $x[n]$ is a wide sense stationary process. This means that $r_x[n,m]$ is independent of n. Therefore, we omit the n in $r_x[n,m]$ and denote $r_x[n,m] = r_x[m]$. The cross-correlation of $z_i[n]$ and $z_j[n]$ is given by

$$r_{z_i,z_j}[n,m] = E(z_i^*[n]z_j[n+m]).$$

Similarly, the cross-correlation of $y_i[n]$ and $y_j[n]$ is given by

$$r_{y_i,y_j}[n,m] = E(y_i^*[n]y_j[n+m]).$$

Since the outputs of the cosets are assumed to be readily available, $r_{y_i,y_j}[n,m]$ can be determined. We plan to estimate $r_x[m]$ by making use of $r_{y_i,y_j}[n,m]$.

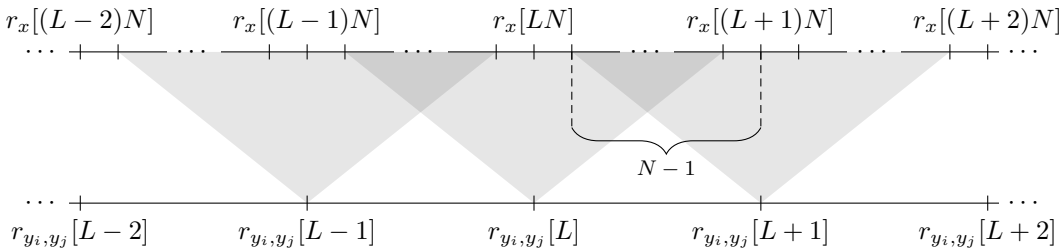


Figure B.4: Dependencies between $r_{y_i,y_j}[m]$ and $r_x[m]$ defined by Equation (B.7)

B.3 Relating the autocorrelation of the input signal

Since $x[n]$ is a wide sense stationary process, Theorem 11.5 of [51] yields that

$$r_{z_i,z_j}[n,m] = (c_{i,j} * r_x)[m] \tag{B.5}$$

where

$$c_{i,j}[m] = \sum_{k=-\infty}^{\infty} c_i^*[k]c_j[k+m]. \tag{B.6}$$

This shows that $r_{z_i,z_j}[n,m]$ is independent of n, we means that we omit the n in $r_{z_i,z_j}[n,m]$ once more. Now the independence of n implies that

$$\begin{aligned} r_{z_i,z_j}[mN] &= E(z_i^*[n]z_j[n+mN]) \\ &= E(z_i^*[nN]z_j[nN+mN]) \\ &= E(y_i^*[n]y_j[n+m]) \\ &= r_{y_i,y_j}[m], \end{aligned}$$

since we defined that $z_i[n] = y_i[nN]$. Therefore

$$r_{y_i,y_j}[m] = (c_{i,j} * r_x)[mN].$$

Consider Equation (B.6). We know that $c_i[n] = 0$ for $n < 0$ and $n > N-1$, which implies that $c_{i,j}[m] = 0$ for $|m| > N-1$, since $c_i[k]c_j^*[k+m] = 0$ for $|m| > N-1$. So

$$\begin{aligned} r_{y_i,y_j}[m] &= \sum_{k=-\infty}^{\infty} r_x[k]c_{i,j}[mN-k] \\ &= \sum_{k=(m-1)N+1}^{(m+1)N-1} r_x[k]c_{i,j}[mN-k]. \end{aligned} \tag{B.7}$$

This is the desired equation which relates the cross-correlation of the outputs of cosets i and j to the autocorrelation of the input signal. We see that every element of $r_{y_i,y_j}[m]$ depends on specific elements of $r_x[m]$. Figure B.4 illustrates these dependencies.

B.4 Estimating the autocorrelation of the input signal

To estimate the autocorrelation of the input signal, we assume that $r_{y_i,y_j}[m] = 0$ for $|m| > L$. This assumption will be discussed at a later moment. If we inspect Figure B.4, we see that $r_{y_i,y_j}[L+1]$ depends on $r_x[LN+1]$ to $r_x[(L+2)N-1]$. Since we have chosen $c_{i,j}[m]$ arbitrarily, $r_{y_i,y_j}[L+1] = 0$ implies that $r_x[LN+1]$ to $r_x[(L+2)N-1]$ must be zero, since $r_{y_i,y_j}[L+1]$ is not necessarily zero otherwise. Consequently, if we evaluate $r_{y_i,y_j}[m] = 0$ for all $|m| > L$ in a similar way, we obtain that $r_x[m] = 0$ for $|m| > LN$.

Since $r_x[m] = 0$ for $|m| > LN$, all that remains is estimating $r_x[m]$ for $|m| \leq LN$. Aggregating Equation (B.7) for $|m| \leq L$ yields that

$$
\begin{aligned}
r_{y_i,y_j}[L] &= \sum_{k=(L-1)N+1}^{(L+1)N-1} r_x[k]c_{i,j}[LN-k] = \sum_{k=(L-1)N+1}^{LN} r_x[k]c_{i,j}[LN-k], \\
r_{y_i,y_j}[L-1] &= \sum_{k=(L-2)N+1}^{LN-1} r_x[k]c_{i,j}[(L-1)N-k], \\
&\vdots \\
r_{y_i,y_j}[-L+1] &= \sum_{k=-LN+1}^{(-L+2)N-1} r_x[k]c_{i,j}[(-L+1)N-k], \\
r_{y_i,y_j}[-L] &= \sum_{k=(-L-1)N+1}^{(-L+1)N-1} r_x[k]c_{i,j}[-LN-k] = \sum_{k=-LN}^{(-L+1)N-1} r_x[k]c_{i,j}[-LN-k].
\end{aligned}
$$

In reducing the limits of the sums in $r_{y_i,y_j}[L]$ and $r_{y_i,y_j}[-L]$ we used that $r_x[m] = 0$ for $|m| > LN$. Although these equations look complicated, they can be written more compactly. To do this, let

$$
\begin{aligned}
\mathbf{r}_x[k] &= \begin{bmatrix} r_x[(k+1)N-1] & \cdots & r_x[kN+1] \end{bmatrix}^T, \\
\mathbf{c}_{i,j}^- &= \begin{bmatrix} c_{i,j}[-N+1] & \cdots & c_{i,j}[-1] \end{bmatrix} \text{ and} \\
\mathbf{c}_{i,j}^+ &= \begin{bmatrix} c_{i,j}[1] & \cdots & c_{i,j}[N-1] \end{bmatrix}.
\end{aligned} \tag{B.8}
$$

If we look carefully, we see that

$$
\begin{aligned}
r_{y_i,y_j}[L] &= r_x[LN]c_{i,j}[0] + \sum_{k=(L-1)N+1}^{LN-1} r_x[k]c_{i,j}[LN-k] \\
&= r_x[LN]c_{i,j}[0] + \mathbf{c}_{i,j}^+ \mathbf{r}_x[L-1].
\end{aligned}
$$

Similarly, we can write

$$
r_{y_i,y_j}[L-1] = \mathbf{c}_{i,j}^- \mathbf{r}_x[L-1] + r_x[(L-1)N]c_{i,j}[0] + \mathbf{c}_{i,j}^+ \mathbf{r}_x[L-2].
$$

The complicated system of equations now reduces to

$$\begin{aligned}
r_{y_i,y_j}[L] &= && r_x[LN]c_{i,j}[0] && + \mathbf{c}^+_{i,j}\mathbf{r}_x[L-1],\\
r_{y_i,y_j}[L-1] &= \mathbf{c}^-_{i,j}\mathbf{r}_x[L-1] && + r_x[(L-1)N]c_{i,j}[0] && + \mathbf{c}^+_{i,j}\mathbf{r}_x[L-2],\\
&\vdots\\
r_{y_i,y_j}[-L+1] &= \mathbf{c}^-_{i,j}\mathbf{r}_x[-L+1] && + r_x[(-L+1)N]c_{i,j}[0] && + \mathbf{c}^+_{i,j}\mathbf{r}_x[-L],\\
r_{y_i,y_j}[-L] &= \mathbf{c}^-_{i,j}\mathbf{r}_x[-L] && + r_x[-LN]c_{i,j}[0], &&
\end{aligned}$$

which can be conveniently written as the matrix equation

$$\begin{bmatrix} r_{y_i,y_j}[L] \\ r_{y_i,y_j}[L-1] \\ \vdots \\ r_{y_i,y_j}[-L+1] \\ r_{y_i,y_j}[-L] \end{bmatrix} = \begin{bmatrix} c_{i,j}[0] & \mathbf{c}^+_{i,j} & & & & & \\ & \mathbf{c}^-_{i,j} & c_{i,j}[0] & \mathbf{c}^+_{i,j} & & & \\ & & & \ddots & & & \\ & & & \mathbf{c}^-_{i,j} & c_{i,j}[0] & \mathbf{c}^+_{i,j} & \\ & & & & & \mathbf{c}^-_{i,j} & c_{i,j}[0] \end{bmatrix} \begin{bmatrix} r_x[LN] \\ r_x[LN-1] \\ \vdots \\ r_x[-LN+1] \\ r_x[-LN] \end{bmatrix}.$$

Denote the previous equation by

$$\mathbf{r}_{y_i,y_j} = \mathbf{R}_{i,j}\mathbf{r}_x. \tag{B.9}$$

This equation relates the cross-correlation of the outputs of cosets i and j to the autocorrelation of the input signal. We can aggregate Equation (B.9) for all combinations of cosets. This yields that

$$\begin{bmatrix} \mathbf{r}_{y_1,y_1} \\ \vdots \\ \mathbf{r}_{y_M,y_M} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{1,1} \\ \vdots \\ \mathbf{R}_{M,M} \end{bmatrix} \mathbf{r}_x.$$

Denote the aggregation of Equation (B.9) by

$$\mathbf{r}_y = \mathbf{R}\mathbf{r}_x. \tag{B.10}$$

Equation (B.10) relates the cross-correlations of all combinations of outputs of cosets to the autocorrelation of the input signal. Note that $\mathbf{R}$ is an $M^2(2L+1) \times 2LN+1$ matrix. If $\mathbf{R}$ has full column rank, then $\mathbf{r}_x$ can be solved for using least-squares, which involves solving the normal equations. These normal equations are given by

$$\mathbf{R}^T\mathbf{r}_y = \mathbf{R}^T\mathbf{R}\mathbf{r}_x. \tag{B.11}$$

Alternatively, the least-squares solution can be obtained by making use of the Moore–Penrose pseudoinverse. Therefore, if $\mathbf{R}$ has full column rank, then

$$\mathbf{R}^\dagger\mathbf{r}_y = \mathbf{r}_x \tag{B.12}$$

where $\mathbf{R}^\dagger$ denotes the Moore-Penrose pseudoinverse of $\mathbf{R}$. We can use Equation (B.11) or Equation (B.12) to estimate the autocorrelation of the input signal if $\mathbf{r}_y$ is known. This concludes the estimation of the autocorrelation of the input signal.

$$\begin{array}{c} 1 \quad 2 \qquad\qquad N \quad N+1 \quad N+2 \\ \left[\begin{array}{cccccc} c_{i,j}[0] & c_{i,j}[1] & \cdots & c_{i,j}[N-1] & 0 & 0 & \cdots \\ 0 & c_{i,j}[-N+1] & \cdots & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & \cdots \\ 0 & 0 & \cdots & 0 & 0 & c_{i,j}[-N+1] & \cdots \end{array}\right. \end{array}$$

Figure B.5: Upper left corner of $\mathbf{R}_{i,j}$. The column numbers are shown and the important elements are highlighted in red.

B.5 Unicity of the estimation

Since Equation (B.11) is used to estimate the autocorrelation of the input signal, we have to make sure that $\mathbf{R}$ has full column rank. We will do this by inspecting the structure of $\mathbf{R}$. Since $\mathbf{R}$ depends on $c_i[n]$ solely, we start by analysing $c_i[n]$.

In the construction of the output of a coset, we saw the output of coset i is an N-decimation of the input signal if $c_i[n] = 1$ for a single n and otherwise zero. This is exactly what we want, since this matches the methods described in Chapter 7. Therefore, assume that $c_i[n] = 1$ for $n = n_i$ and otherwise zero. Now consider Equation (B.6). Note that $c_{i,j}[m] = 1$ if and only if $c_i^*[k] = 1$ and $c_j[k+m] = 1$, which requires that $k = n_i$ and $k + m = n_j$. Equivalently, $c_{i,j}[m] = 1$ if and only if $m = n_j - n_i$. This means that $c_{i,j}[m]$ consists of a single one. Now consider Equation (B.9). Since $c_{i,j}[m]$ consists of a single one, two columns of $\mathbf{R}_{i,j}$ are linearly independent if they are nonzero. Therefore, if all columns of $\mathbf{R}$ are nonzero, then all columns are linearly independent and $\mathbf{R}$ has full column rank.

Suppose that the first N columns of $\mathbf{R}$ are nonzero. Consider Equation (B.9) in conjunction with Figure B.5. We see that the structure of $\mathbf{R}$ repeats itself in a diagonal way every N samples. Therefore, if the first N columns are nonzero, all columns are nonzero and thus $\mathbf{R}$ has full column rank. The problem now reduces to choosing $c_i[n]$ such that the first N columns of $\mathbf{R}$ are nonzero.

Consider $c_i[n]$ and $c_j[n]$. This means that $c_{i,j}[m]$ and $c_{j,i}[m]$ exist. We have seen that $c_{i,j}[m] = 1$ if and only if $m = n_j - n_i$ and $c_{j,i}[m] = 1$ if and only if $m = n_i - n_j$. We make the following observations.

1. If $n_j - n_i \geq 0$, then Figure B.5 shows that the $n_j - n_i + 1$'th column is nonzero. Similarly, if $n_i - n_j \geq 0$, then the $n_i - n_j + 1$'th column is nonzero. Either way, the $|n_i - n_j| + 1$'th column is nonzero.
2. If $n_j - n_i \leq 0$, then Figure B.5 shows that the $N - (n_i - n_j) + 1$'th column is nonzero. Similarly, if $n_i - n_j \leq 0$, then the $N - (n_j - n_i) + 1$'th column is nonzero. Either way, the $N - |n_i - n_j| + 1$'th column is nonzero.

We require that columns 1 to N are nonzero. This implies the following statement. If $|n_i - n_j|$ and $N - |n_i - n_j|$ for all combinations of i and j make up 0 to $N - 1$, then all columns of $\mathbf{R}$ are nonzero and $\mathbf{R}$ has full column rank. This restriction will be further discussed in 9.

C | Construction of the matrices

The operation of `toeplitz` and `tril` is illustrated with two examples.

$$\text{toeplitz}\left(\begin{bmatrix}1 & 2 & 3 & 4\end{bmatrix}\right) = \begin{bmatrix}1 & 2 & 3 & 4\\ 2 & 1 & 2 & 3\\ 3 & 2 & 1 & 2\\ 4 & 3 & 2 & 1\end{bmatrix},$$

$$\text{tril}\left(\begin{bmatrix}1 & 2 & 3 & 4\\ 2 & 1 & 2 & 3\\ 3 & 2 & 1 & 2\\ 4 & 3 & 2 & 1\end{bmatrix}\right) = \begin{bmatrix}1 & 0 & 0 & 0\\ 2 & 1 & 0 & 0\\ 3 & 2 & 1 & 0\\ 4 & 3 & 2 & 1\end{bmatrix}.$$

To illustrate the way $\mathbf{R}_{i,j}$ can be generated efficiently, we consider an example. Suppose that $N = 2$ and $L = 1$. Then

$$\mathbf{R}_{i,j} = \begin{bmatrix}c_{i,j}[0] & c_{i,j}[1] & 0 & 0 & 0\\ 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0\\ 0 & 0 & 0 & c_{i,j}[-1] & c_{i,j}[0]\end{bmatrix}.$$

Now notice that

$$\text{tril}\left[\text{toeplitz}\left(\begin{bmatrix}c_{i,j}[1] & c_{i,j}[0] & c_{i,j}[-1] & 0 & 0 & 0\end{bmatrix}\right)\right] =$$
$$\begin{bmatrix}c_{i,j}[1] & 0 & 0 & 0 & 0 & 0\\ c_{i,j}[0] & c_{i,j}[1] & 0 & 0 & 0 & 0\\ c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0 & 0 & 0\\ 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0 & 0\\ 0 & 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0\\ 0 & 0 & 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1]\end{bmatrix}.$$

We can construct $\mathbf{R}_{i,j}$ by omitting several rows and columns. First omit the first $N-1$ rows and last $N-1$ columns. Then

$$\begin{bmatrix}c_{i,j}[1] & 0 & 0 & 0 & 0 & 0\\ c_{i,j}[0] & c_{i,j}[1] & 0 & 0 & 0 & 0\\ c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0 & 0 & 0\\ 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0 & 0\\ 0 & 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0\\ 0 & 0 & 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1]\end{bmatrix}$$

remains. The red elements represent the omitted elements. Now keep rows $1, N+1, \ldots, 2LN+1$. Then

$$\begin{bmatrix} c_{i,j}[0] & c_{i,j}[1] & 0 & 0 & 0 \\ c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0 & 0 \\ 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] & 0 \\ 0 & 0 & c_{i,j}[-1] & c_{i,j}[0] & c_{i,j}[1] \\ 0 & 0 & 0 & c_{i,j}[-1] & c_{i,j}[0] \end{bmatrix}$$

remains, which is exactly $\mathbf{R}_{i,j}$.

D | Circular sparse ruler problem

The circular sparse ruler problem was introduced in Chapter 8 and reformulated in Chapter 9. Consider a circle with N marks on it. Every mark is occupied by a sampler. Time on this circle passes as the hand of a clock. By continuity of the circle, the lag d is estimated if the two marks are d apart. The circle is illustrated in Figure D.1. Here $N = 7$ and the positions 1, 2 and 3 are marked. The circle is a solution to the circular sparse ruler problem, since the distances between the marks make up $0, \ldots, N-1$.

Finding solutions to the circular sparse ruler problem where the amount of marks is minimised is relevant to us, since a minimal amount of marks is equivalent to using a minimal amount of samplers for a given N. We call these solutions minimal circular sparse ruler solutions.

[2] proposes a method that uses the so called linear sparse ruler problem to generate solution for the circular sparse ruler problem. The linear sparse ruler problem is similar to the circular sparse ruler problem. If one disconnects the end of the circle in Figure D.1, but still requires that the marks make up the distances $0, \ldots, N-1$, one obtains the linear sparse ruler problem. This is illustrated in Figure D.2.

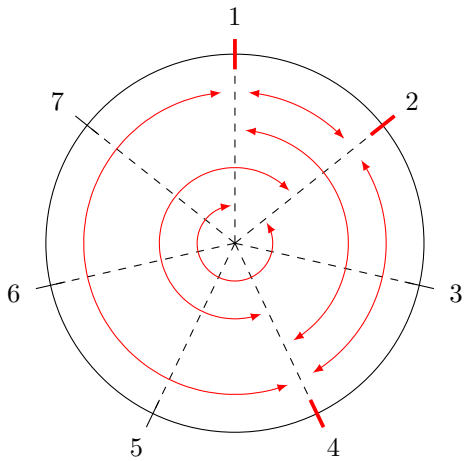


Figure D.1: Circular sparse ruler problem for $N = 7$

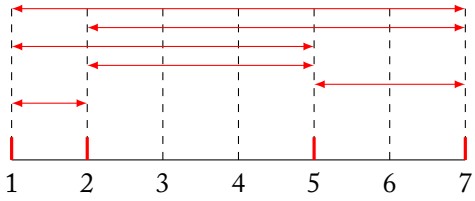


Figure D.2: Linear sparse ruler for $N = 7$ and $M = 4$

The proposed method in [2] uses the fact that a linear sparse solution for a certain N can be turned into circular sparse ruler solutions for $2N - 2$ when N is even and $2N-1$ when N is odd. To see this, simply note that every distance d between marks on a circle also yields a distance $N - d$. The linear sparse solution in Figure D.2 is therefore also a circular sparse ruler solution for $N = 13$. This is illustrated in Figure D.3.

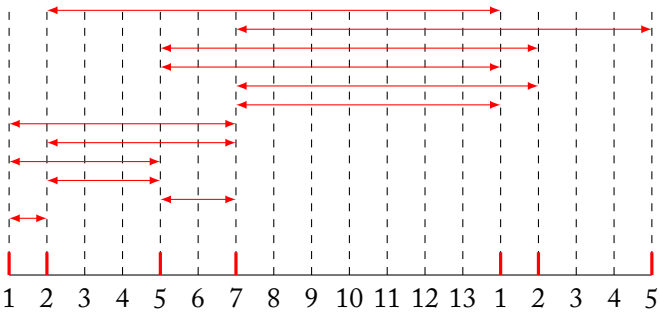


Figure D.3: Circular sparse solution for $N = 13$ generated from the linear sparse ruler solution for $N = 7$. The numbers on the axis repeat since this is a circular sparse ruler problem.

The generation approach is advantageous, since solutions of linear sparse ruler problem can be turned into circular sparse ruler solutions. Therefore better linear sparse ruler solutions will also result in better circular sparse ruler solutions. Because linear sparse ruler is a well-known problem, good solutions are already calculated. However, solutions generated for the circular sparse ruler problem are not necessarily minimal. Finding minimal solutions is by no means trivial and requires clever algorithms. It is also interesting to look at the structure of the problem to decrease the solution space.

D.1 Structure of the circular sparse ruler problem

Optimal circular sparse rulers form interesting solutions to the circular sparse ruler problem. They are circular sparse ruler solutions to configurations with M samplers such that N is as high as possible. Optimal solutions ensure the highest performance possible for a given amount of samplers. In similar fashion, one can define optimal solutions to the linear sparse ruler problem.

M	N	**Marks at positions**
2	3	$1, 2$
3	7	$1, 2, 4$
4	13	$1, 2, 5, 7$
5	21	$1, 2, 7, 9, 19$
6	31	$1, 2, 5, 7, 14, 22$
7	39	$1, 2, 5, 7, 14, 21, 31$
8	51	$1, 2, 8, 10, 13, 32, 36, 46$
9	65	$1, 2, 5, 7, 15, 33, 43, 51, 54$

Table D.1: Several solutions to the circular sparse ruler problem

THEOREM 1

(*Upper bound on circular sparse ruler solutions.*) Consider a configuration with M samplers. Then $N \leq M(M-1)+1$.

PROOF OF THEOREM 1

There are M^2 combinations of samplers. However, M of them result in a combination with themselves, which yields an estimation of lag 0. Therefore, only $M^2 - M + 1 = M(M-1)+1$ different lags can be estimated. The estimated lags must make up $0, \ldots, N-1$, which are N different lags. Therefore $N \leq M(M-1)+1$. □

D.2 Results

We have used integer linear programming to calculate circular sparse ruler solutions. This program is explained and derived in Appendix E. The solutions are shown in Table D.1.

Figure D.4 compares the upper bound from Theorem 1 to minimal linear sparse ruler solutions and minimal circular sparse ruler solutions. One can clearly see that for high M, minimal solutions to the circular sparse ruler problem have higher N.

D.3 Collaborative sampling

Collaborative sampling also requires that the circular sparse ruler problem is satisfied. In this case, however, the estimation of the lags is split across two physically separate devices. This is shown in Figure D.5. The red arrows indicate the lags that are produced by device one, and the blue arrows indicate the lags that are produced by device two. Note that together they make all lags available. [7] investigates ways to come up with optimal set-ups for different amounts of devices.

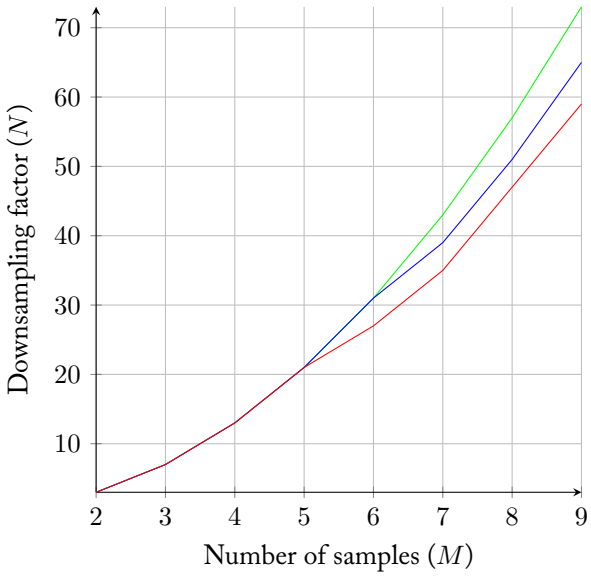


Figure D.4: Solutions of the circular and linear sparse ruler problem. Minimal solutions to linear sparse ruler are shown in red, minimal solutions to the circular sparse ruler problem are shown in blue and the upper bound from Theorem 1 is shown in green.

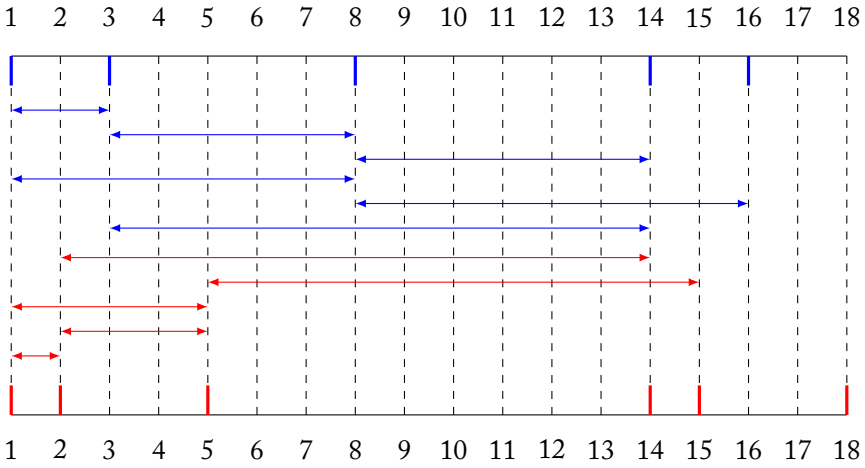


Figure D.5: Collaborative sampling with $N = 13$ and $M = 4$

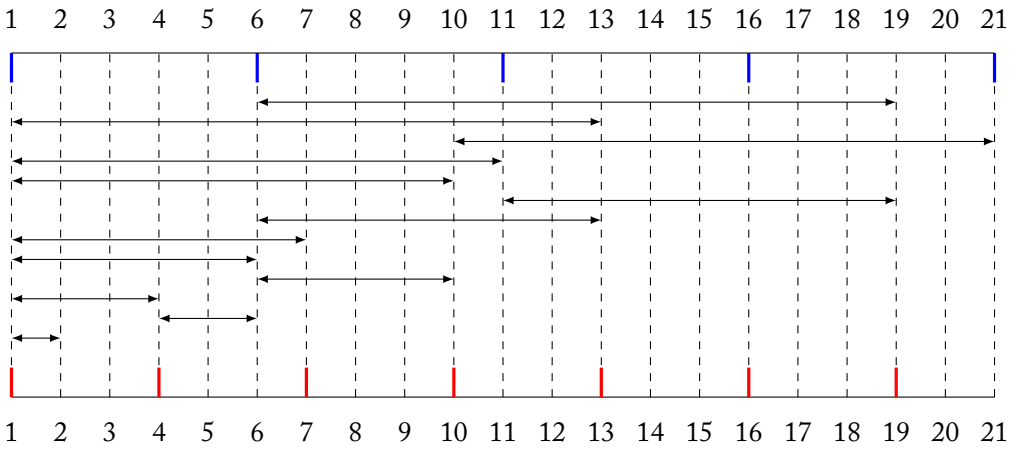


Figure D.6: Coprime sampling with $a = 3$ and $b = 5$

D.4 Coprime sampling

Similarly, coprime sampling also requires that the circular sparse ruler problem is satisfied. A configuration for coprime sampling is depicted in Figure D.6.

We see that the sampling periods a and b should be chosen carefully such that all lags are estimated. [8] shows that choosing a and b coprime yields that all lags are estimated. Remember that a and b are coprime if the only positive integer that evenly divides both of them is one, hence the name coprime.

E | Generating solutions to the circular sparse ruler problem

We reintroduce the concept of a ruler. Consider a ruler $\mathbf{r}$ of length N. Then $\mathbf{r} \in \{0,1\}^N$. Also, $(\mathbf{r})_i = 1$ if the element at position i is marked, and $(\mathbf{r})_i = 0$ if the elements at position is not marked. If an element at position i is marked, then a coset samples the i'th sample of every group of N samples of the input signal. We will formulate the minimal circular ruler problem as an binary integer linear program.

A ruler $\mathbf{r}$ contains a distance d if there exist i and j such that $(\mathbf{r})_i = 1$, $(\mathbf{r})_j = 1$ and $|i-j| = d$ or $N - |i-j| = d$. The ruler $\mathbf{r}$ is a circular ruler if it contains distances $1, \ldots, N-1$. A circular ruler $\mathbf{r}$ is minimal if and only if $||\mathbf{r}||_1$ is minimal.

Let D_d be the set consisting of the vectors $\mathbf{d} \in \{0,1\}^N$ such that $(\mathbf{d})_i = 1$ and $(\mathbf{d})_{i+d} = 1$ where $i = 1, \ldots, N-d$ or $(\mathbf{d})_i = 1$ and $(\mathbf{d})_{i+d-N}$ where $i = N-d+1, \ldots, N-1$.

THEOREM 2 Let $\mathbf{r} \in \{0,1\}^N$ be a ruler. Then $\mathbf{r}$ contains the distance d if $\mathbf{d}^T\mathbf{r} = 2$ for any $\mathbf{d} \in D_d$.

PROOF OF THEOREM 2 Assume that $\mathbf{d}^T\mathbf{r} = 2$. Then there exist i and j such that $(\mathbf{d})_i(\mathbf{r})_i + (\mathbf{d})_j(\mathbf{r})_j = 2$. This implies that $(\mathbf{r})_i = 1$ and $(\mathbf{r})_j = 1$. Since $\mathbf{d} \in D_d$, either $|i-j| = d$ or $|i-j| = N-d$. In both cases, $\mathbf{r}$ contains the distance d. □

Let $\mathbf{D}_d$ denote the $|D_d| \times N$ matrix which contains all vectors in D_d as rows.

THEOREM 3 Let $\mathbf{r} \in \{0,1\}^N$ be a ruler and let $\mathbf{y} \in \{0,1\}^{|D_d|}$. Then $\mathbf{r}$ contains the distance d at least once if $\mathbf{D}_d\mathbf{r} \geq 2\mathbf{y}$ such that $||\mathbf{y}||_1 = 1$.

PROOF OF THEOREM 3 Assume that $\mathbf{D}_d\mathbf{r} \geq 2\mathbf{y}$ such that $||\mathbf{y}||_1 = 1$. Then there is one and only one i such that $(\mathbf{y})_i = 1$. Therefore there exists a $\mathbf{d} \in D_d$ such that $\mathbf{d}^T\mathbf{r} \geq 2(\mathbf{y})_i = 2$. Since $||\mathbf{d}||_1 = 2$, the equality holds and thus $\mathbf{r}$ contains the distance d. Now consider

$\mathbf{d}' \in D_d$ such that $\mathbf{d}' \neq \mathbf{d}$. Then $\mathbf{d}^T\mathbf{r} \geq 0$, which holds for any $\mathbf{r}$. □

Consider the binary integer linear program

$$\begin{aligned}
&\text{minimise } \mathbf{1}_N^T\mathbf{r} \\
&\text{such that } \begin{bmatrix} \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_{N-1} \end{bmatrix} \mathbf{r} \geq 2 \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_2 \end{bmatrix}, \begin{bmatrix} \mathbf{1}_{|D_1|}^T & & \\ & \ddots & \\ & & \mathbf{1}_{|D_{N-1}|}^T \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{N-1} \end{bmatrix} = \mathbf{1}_{N-1}, \\
&\qquad \mathbf{r} \in \{0,1\}^N \text{ and } \mathbf{y}_i \in \{0,1\}^{|D_i|} \text{ for } i = 1, \ldots, N-1.
\end{aligned}$$

Then, by the definition of a minimal circular ruler and Theorem 3, this program yields a minimal circular ruler of length N.

F | Appendix on detection

This appendix will focus on the detailed theory behind the detectors as described in Chapter 10. First a set of definitions is given which will be used in subsequent explanations and derivations.

F.1 Preliminaries

The definitions given in this section are used in the derivations later on and are based on [51], [52].

DEFINITION 1

(*Complex Gaussian Random Variable.*) Given the complex Gaussian random variable $Z = X + jY$, then Z is said to be complex Gaussian distributed if X and Y are jointly-Gaussian distributed. We denote $Z \sim \mathcal{CN}(\mu, \Gamma, C)$.

A complex Gaussian random variable is characterised by three parameters:

1. $\mu = E(Z)$,
2. $\Gamma = E[(Z - \mu)(Z^* - \mu^*)^T]$ and
3. $C = E[(Z - \mu)(Z - \mu)^T]$.

DEFINITION 2

(*Circular Symmetric Complex Gaussian Random Variable.*) Given a complex Gaussian random variable $Z = X + jY$, then Z is circular symmetric complex Gaussian distributed if Z has the same distribution as $Ze^{j\theta}$ with θ real. A circular complex Gaussian random variable Z is referred to as $Z \sim \mathcal{CN}(0, \sigma^2)$.

Since

$$E(Z) = E(e^{j\theta}Z) = e^{j\theta}E(Z)$$

it follows that Z has an expectation value of zero. To see why let$E(Z) = a + bj$ and notice that $e^{j\theta} \neq 0$. Then if $e^{j\theta}E(Z) = E(z)$ holds, then so does $(a+bj) = e^{j\theta}(a+$

bj). Now it follows that $E(Z)$ cannot be anything else than zero. Furthermore, since

$$E(ZZ) = E(e^{j\theta}Ze^{j}\theta Z) = e^{2j\theta}E(Z^2)$$

it follows that $\text{Var}\,(X) = \text{Var}\,(Y)$ (by an analog argument as for the expectation value). That is, X and Y are independent Gaussian distributed with zero mean and equal variance. It now follows that $\sigma^2 = E(Z\overline{Z}) = \text{Var}\,(X) + \text{Var}\,(Y)$.

DEFINITION 3 (*Circular Symmetric Complex Gaussian Random Vector.*) Given the random vector $\mathbf{Z} \in \mathbb{C}^N$, then $\mathbf{Z} = \mathbf{X} + j\mathbf{Y}$ is a circular symmetric complex random vector if $\mathbf{X}$ and $\mathbf{Y}$ are jointly Gaussian distributed and $\mathbf{Z}$ has the same distribution as $e^{j\theta}\mathbf{Z}$ with θ real.

Similar to a circular symmetric complex Gaussian variable, $E(\mathbf{Z}) = \mathbf{0}$ and $E(\mathbf{Z}\mathbf{Z}^T) = \mathbf{0}$[52]. A circular symmetric complex Gaussian random vector $\mathbf{Z}$ is referred to as $\mathbf{Z} \sim \mathcal{CN}(\mathbf{0}, \mathbf{\Gamma}, \mathbf{0})$ with $\mathbf{\Gamma} = E(\mathbf{Z}\mathbf{Z}^*)$. For the standard circular symmetric complex Gaussian random vector, $\mathbf{\Gamma} = \mathbf{I}$. The probability density function of a circular symmetric complex Gaussian vector $\mathbf{Z} \in \mathbb{C}^N$ is given by

$$\frac{1}{\pi^N \det(\mathbf{\Gamma})} \exp\left(-(\mathbf{z}^T)^* \mathbf{\Gamma}^{-1}\mathbf{z}\right).$$

DEFINITION 4 (*Chi-square distribution for complex random variables.*) Given a random vector $\mathbf{Z} \in \mathbf{C}^N$, with $Z_i \sim \mathcal{CN}(\mathbf{0}, 2\mathbf{I}, \mathbf{0})$ then the random variable $\mathbf{X}$ defined as

$$\mathbf{X} = \sum_{n=1}^{N} \left|(\mathbf{Z})_n\right|^2$$

follows a chi-square distribution with $2N$ degrees of freedom, denoted by $\mathbf{X} \sim \chi^2_{2N}$. $E[X] = 2N$ and $\text{Var}[X] = 4N$.

DEFINITION 5 (*Likelihood function under Hypothesis.*) Given a hypothesis $\mathcal{H}$ and a realisation $\mathbf{x}$ of a random variable $\mathbf{X}$, then $L_{\mathbf{X}|\mathcal{H}}(\mathbf{x})$ denotes the likelihood function of $\mathbf{x}$ given $\mathcal{H}$.

DEFINITION 6 (*Neyman-Pearson Test.*) Given a continuous random vector $\mathbf{X}$, and two hypotheses $\mathcal{H}_0$ and $\mathcal{H}_1$, the Neyman-Pearson test rejects that the realization $\mathbf{x}$ of $\mathbf{X}$, has been produced under $\mathcal{H}_0$ in favor of $\mathcal{H}_1$ when

$$\Lambda(\mathbf{x}) = \frac{L_{\mathbf{X}|\mathcal{H}_0}(\mathbf{x})}{L_{\mathbf{X}|\mathcal{H}_1}(\mathbf{x})} \leq \gamma.$$

where γ, the decision threshold, is chosen such that $P[\Lambda(\mathbf{x} \leq \gamma)|\mathcal{H}_1]$ is maximized subject to $P[(]\Lambda(\mathbf{x}) \leq \gamma)|\mathcal{H}_0] = P_{fa}$, where P_{fa} denotes the false alarm probability.

DEFINITION 7 (*Covariance Matrix.*) Given a vector $\mathbf{x}$, its covariance matrix is defined as $C_x = E(\mathbf{x}\mathbf{x}^H) - E(\mathbf{x})E(\mathbf{x}^H)$

F.2 Energy detector

Conventional Energy detector

This section will give a derivation of the conventional energy detector based on [26]. Let $x[n]$ denote the received signal. The conventional energy detection algorithm must decide between to hypotheses:

$$\mathcal{H}_0 : x[n] = w[n], \tag{F.1}$$
$$\mathcal{H}_1 : x[n] = s[n] + w[n], \tag{F.2}$$

in which $w[n]$ denotes additive circular symmetric complex Gaussian noise and $s[n]$ denotes a signal as transmitted by a primary user.

We assume that the noise samples are independently identically zero mean circular symmetric complex Gaussian distributed, that is $w[n] \sim \mathcal{CN}(0, \sigma_n^2)$. We furthermore assume that the samples of the signal $s[n]$ can be modelled independently as circular symmetric complex Gaussian $\mathcal{CN}(0, \sigma_s^2)$. So

$$x[n] \sim \begin{cases} \mathcal{CN}(0, \sigma_n^2) & \text{if } \mathcal{H}_0, \\ \mathcal{CN}[0, (\sigma_s^2 + \sigma_n^2)] & \text{if } \mathcal{H}_1. \end{cases}$$

Let

$$\mathbf{x} = \begin{bmatrix} x[0] & x[1] & \dots & x[N-1] \end{bmatrix}^T$$

denote a vector containing N samples of the signal $x[n]$. Then the likelihood function of $\mathbf{x}$ denoted as $L(\mathbf{x})$ is given by

$$\begin{aligned} L(\mathbf{x}) &= \prod_{i=1}^{N} f_{(\mathbf{x})_i} \\ &= \begin{cases} 1/\sigma_n^2 \pi^N \cdot \exp(-\bar{\mathbf{x}}' \sigma_n^{-2} \mathbf{I}\mathbf{x}) & \text{if } \mathcal{H}_0, \\ 1/(\sigma_n^2 + \sigma_s^2)\pi^N \cdot \exp[-\bar{\mathbf{x}}'(\sigma_n^2 + \sigma_s^2)^{-1}\mathbf{I}\mathbf{x}] & \text{if } \mathcal{H}_1 \end{cases} \end{aligned}$$

where $f_{(\mathbf{x})_i}$ denotes the probability density function of element i in $\mathbf{x}$. The test statistic $\Lambda(\mathbf{x})$ as used in the Neyman Pearson test is then given by:

$$\Lambda(\mathbf{x}) = \frac{1/\sigma_n^2 \pi^N \cdot \exp(-\bar{\mathbf{x}}' \sigma_n^{-2} \mathbf{I}\mathbf{x})}{1/(\sigma_n^2 + \sigma_s^2)\pi^N \cdot \exp[-\bar{\mathbf{x}}'(\sigma_n^2 + \sigma_s^2)^{-1}\mathbf{I}\mathbf{x}]}.$$

By taking the logarithm of $\Lambda(x)$ we obtain a log-likelihood Ratio test statistic $\Lambda'(x)$, given by

$$\begin{aligned}\Lambda(\mathbf{x})' &= \log\left\{\frac{1/\sigma_n^2\pi^N \cdot \exp(-\bar{\mathbf{x}}'\sigma_n^{-2}\mathbf{I}\mathbf{x})}{1/(\sigma_n^2+\sigma_s^2)\pi^N \cdot \exp[-\bar{\mathbf{x}}'(\sigma_n^2+\sigma_s^2)^{-1}\mathbf{I}\mathbf{x}]}\right\} \\ &= -\log(\sigma_n^2) + \log(\sigma_n^2+\sigma_s^2) - \left(\frac{1}{\sigma_n^2} - \frac{1}{\sigma_n^2+\sigma_s^2}\right)\sum_{i=0}^{N-1}|x[i]|^2.\end{aligned} \tag{F.3}$$

Observing that the constants σ_n and σ_s do not depend on value of the samples, the test statistic[1]

$$\Lambda''(\mathbf{x}) = -\sum_{n=0}^{N-1}|x[n]|^2$$

is proportional to $\Lambda'(x)$. If we negate Λ'' we get same test statistic as denoted by Λ in Section 10.3. Do note that this negation is effectively a division in the original Neyman-Pearson test and therefore if $-\Lambda'' > \gamma$, the detector decides that $\mathcal{H}_1$ is the true hypothesis.

Threshold

In this section we will determine the threshold γ for the energy detector test statistic $-\Lambda''(\mathbf{x})$. Under $\mathcal{H}_0$ we have that $\Lambda''(\mathbf{x})$ is the sum of $2N$ independent zero mean Gaussian distributed variables with variance $\sigma_n^2/2$. To see why, remember that $\mathbf{x}$ is circular symmetric complex Gaussian and therefore for the element $x_i \in \mathbf{x}$ the following applies $|x_i|^2 = x_i \cdot x_i^* = a^2 + b^2$ where $a = \mathrm{Re}(x_i)$ and $b = \Im(x_i)$. So

$$\frac{-2\Lambda''(\mathbf{x})}{\sigma_n^2} \sim \chi^2_{2N}. \tag{F.4}$$

Therefore, the false alarm probability p_{fa} is given by [53]

$$\begin{aligned}p_{fa} &= P(-\Lambda''(\mathbf{x}) > \gamma) \\ &= 1 - F_{2N}(2\gamma/\sigma_n^2)\end{aligned}$$

with F_{2N} the cumulative distribution function of a chi square distribution with $2N$ degrees of freedom.

If N is large enough, we can approximate the test statistics' distribution by a Gaussian distribution as it it is the sum of $2N$ independent identically distributed random variables. By the central limit theorem,

[1] observe that $\frac{1}{\sigma_n^2} \geq \frac{1}{\sigma_n^2+\sigma_s^2}$

$$F_{2N} \approx 1 - Q\left(\frac{2\Lambda''(x)/\sigma_n^2 - 2N}{2\sqrt{N}}\right).$$

P_{fa} can then be approximated as

$$P_{fa} \approx Q\left(\frac{2\gamma/\sigma_n^2 - 2N}{2\sqrt{N}}\right). \tag{F.5}$$

Given a desired P_{fa}, the threshold γ is given by

$$\gamma = [Q^{-1}\left(p_{fa}\right)\sqrt{N} + N]\sigma_n^2. \tag{F.6}$$

F.3 Energy detector using the reconstructed autocorrelation

In this section we will derive how γ_k as introduced in Section 10.3 can be calculated. This derivation is largely based on [2] and is heavily dependent on the derivation in Appendix B.

We assume that if our signal $x[n]$ purely contains noise, then $x[n] \sim \mathcal{CN}(0, \sigma_n^2)$. Let

$$\hat{\mathbf{r}}_x = \begin{bmatrix}\hat{r}[-LN] & \cdots & \hat{r}[0] & \cdots & \hat{r}[LN]\end{bmatrix}^T$$

denote the vector containing the estimated elements of the autocorrelation. By Equation (B.12) we know that the estimator $\hat{\mathbf{r}}_x$ of $\mathbf{r}_x$ is given as

$$\hat{\mathbf{r}}_x = \mathbf{R}^\dagger \hat{\mathbf{r}}_y.$$

where the elements of the estimator $\hat{\mathbf{r}}_y$ of $\mathbf{r}_y$ are given by Equation (8.1).

Let $\mathbf{F}$ denote the $(2LN+1) \times (2LN+1)$ discrete Fourier transform matrix, then $\hat{\mathbf{s}}_x$, defined as $\hat{\mathbf{s}}_x = \mathbf{F}\hat{\mathbf{r}}_x$ is an estimator of the power spectral density $\mathcal{P}(\omega)$ at frequencies $\omega = 2\pi k/(2LN+1)$ where $0 \leq k \leq 2LN$.

In [2] it is derived that when KLN is large, then the elements of $\mathbf{s}_x$ are approximately Gaussian distributed. Let $\hat{s}_k$ denote the k'th element of $\hat{\mathbf{s}}_x$, then we approximate its distribution by the Gaussian $\mathcal{N}(\mu_k, \sigma_k^2)$.

The threshold γ_k is then given by

$$\gamma_k = Q^{-1}(p_{fa})\sigma_k + \mu_k. \tag{F.7}$$

Note that μ_k is equal to the k'th element of the vector

$$E\left(\hat{\mathbf{s}}_x\right) = \mathbf{F}\mathbf{R}^\dagger E(\mathbf{r}_y).$$

Making use of the following relation as derived in [2], giving the elements of $E(\mathbf{r}_y)$

$$\begin{aligned} E(\mathbf{r}_{y_i,y_j}[k]) &= E[(c_{i,j} * r_x)(kN)] \qquad \text{Equation (B.5)} \\ &= c_{i,j}\delta[k] \end{aligned}$$

we can calculate μ_k for $1 \leq k \leq 2LN + 1$.

To calculate σ_k we note that the k'th element on the diagonal of the covariance matrix of $\hat{\mathbf{s}}_x$ contains the variance σ_k^2. Let $\mathbf{C}_{\hat{s}_x}$ denote the covariance matrix of $\hat{\mathbf{s}}_x$. We can compute $\mathbf{C}_{\hat{s}_x}$ by

$$\begin{aligned} \mathbf{C}_{\hat{s}_x} &= E(\hat{\mathbf{s}}_x\hat{\mathbf{s}}_x^H) - E(\hat{\mathbf{s}})E(\hat{\mathbf{s}}^H) \\ &= E[(\mathbf{F}\mathbf{R}^\dagger\hat{\mathbf{r}}_y)(\mathbf{F}\mathbf{R}^\dagger\hat{\mathbf{r}}_y)^H] - E(\mathbf{F}\mathbf{R}^\dagger\hat{\mathbf{r}}_y)E[(\mathbf{F}\mathbf{R}^\dagger\hat{\mathbf{r}}_y)^H] \\ &= \mathbf{F}\mathbf{R}^\dagger E(\mathbf{r}_y\mathbf{r}_y^H)\mathbf{R}^{\dagger H}\mathbf{F}^H - \mathbf{F}\mathbf{R}^\dagger E(\mathbf{r}_y)E(\mathbf{r}_y^H)\mathbf{R}^{\dagger H}\mathbf{F}^H \\ &= \mathbf{F}\mathbf{R}^\dagger\mathbf{C}_{r_y}\mathbf{R}^{\dagger H}\mathbf{F}^H. \end{aligned} \tag{F.8}$$

Let $\mathbf{C}_{\hat{r}_y}$ denote the covariance matrix of of $\hat{\mathbf{r}}_y$, with its elements given by

$$\text{Cov}(\hat{r}_{y_t,y_u}[i], \hat{r}_{y_v,y_w}[j]). \tag{F.9}$$

As stated in [2], under the assumption that the elements of $x[n]$ are independent identically circular complex Gaussian distributed, the expression for the covariance in Equation (F.9) simplifies to

$$\text{Cov}(\hat{r}_{y_t,y_u}[i], \hat{r}_{y_v,y_w}[j]) = \frac{\sigma_n^4 c_{t,v}[0]c_{u,w}^*[0]\delta\left[j-i\right]}{KL - |i|}. \tag{F.10}$$

Now that we have described how to calculate μ_k and σ_k, Equation (F.7) can be evaluated.

F.4 Covariance Absolute Value detection

Explanation

Let L samples of the signal $x[n]$ be collected in the vector

$$\mathbf{x} = \begin{bmatrix} x[n] & x[n-1] & \cdots & x[n-L+1] \end{bmatrix}^T.$$

Furthermore let

$$\mathbf{C} = E[(\mathbf{x} - \mu)(\mathbf{x} - \mu)^H]$$

denote the covariance of $\mathbf{x}$ with $\mu = E(\mathbf{x})$. In the case that $E(\mathbf{x}) = 0$, like for noise or most communication signals, then $\mathbf{C}$ can be simplified to

$$\begin{aligned}\mathbf{C}_x &= E(\mathbf{x}\mathbf{x}^H) \\ &= \begin{bmatrix} E\left(x[n]x^*[n]\right) & E\left(x[n]x^*[n-1]\right) & \dots & E\left(x[n]x^*[n-L+1]\right) \\ E\left(x[n-1]x^*[n]\right) & E\left(x[n-1]x^*[n-1]\right) & \dots & E\left(x[n-1]x^*[n-L+1]\right) \\ \vdots & \vdots & \ddots & \vdots \\ E\left(x[n-L+1]x^*[n]\right) & E\left(x[n-L+1]x^*[n-1]\right) & \dots & E\left(x[n-L+1]x^*[n-L+1]\right) \end{bmatrix}.\end{aligned}$$

Under the assumption that $x[n]$ is a wide-sense stationary signal, we can simplify $\mathbf{C}$ even further.

$$\begin{aligned}\mathbf{C}_x &= E(\mathbf{x}\mathbf{x}^H) \\ &= \begin{bmatrix} r_x[0] & r_x[1] & \dots & r_x[L-1] \\ r_x[1] & r_x[0] & \dots & r_x[L-2] \\ \vdots & \vdots & \ddots & \vdots \\ r_x[L-1] & r_x[L-2] & \dots & r_x[0] \end{bmatrix}.\end{aligned}$$

Note how $\mathbf{C}_x$ is symmetric and has a Toeplitz structure. This corresponds to the first block in Figure 10.4. As the autocorrelation function of white noise is a delta function, it means that if $x[n]$ is white noise with variance σ_n^2, then $\mathbf{C}_x = \sigma_n^2\mathbf{I}$. If the signal $x[n]$ is not equal to noise, then its autocorrelation function is not equal to a delta function which results in $\mathbf{C}_x$ having nonzero off diagonal elements.

Covariance Absolute Value method detection uses a measure of this 'diagonality' of $\mathbf{C}_x$ as test statistic Λ. This measure Λ is defined as

$$\begin{aligned}\Lambda &= \frac{T_1}{T_2} \\ &= \frac{\sum_{n=1}^{L}\sum_{m=1}^{L}\left|(\mathbf{C}_x)_{nm}\right|}{\sum_{k=1}^{L}\left|(\mathbf{C}_x)_{kk}\right|}\end{aligned} \tag{F.11}$$

where $T_1 = \sum_{n=1}^{L}\sum_{m=1}^{L}|(\mathbf{C}_x)_{nm}|/L$ and $T_2 = \sum_{k=1}^{L}|(\mathbf{C}_x)_{kk}|/L$. This test statistic can be computed by first taking the absolute value of $\mathbf{C}_x$. This is then followed by summing all the elements of the resulting matrix, T_1, and computing the trace of that matrix, T_2. Upon diving those two results one obtains the test statistic Λ. This process is depicted in Figure 10.4.

In practice one estimates the matrix $\mathbf{C}_x$ by using a limited amount of samples N to estimate $r_x[n]$. The threshold given a desired false alarm probability p_{fa} is derived in [14] to be

$$\gamma = \frac{1 + (L-1)\sqrt{2/N\pi}}{1 - Q^{-1}(p_{fa})\sqrt{2/N}}.$$

This threshold, however, does not apply to the estimated autocorrelation of our reconstruction method. [14] assumes that N samples are used to estimate *each* element of the autocorrelation in $\mathbf{C}_x$ as:

$$r_x[m] = \frac{1}{N}\sum_{n=0}^{N-1} x[n]x^*[n-m] \qquad m = 0, 1, \ldots, L-1. \tag{F.12}$$

Our reconstruction, first of all does not use Equation (F.12) to estimate the autocorrelation. Second, as [14] uses the variance of $r_x[m]$ when estimated by Equation (F.12), in its threshold derivation we know that this threshold is not applicable to our reconstructed estimate, as the variance of the estimated $r_x[m]$ is not necessarily constant (to see why consider Equation (F.8) without multiplication by **F**) and the same as in Equation (F.12).

G | Jammer

Introduction

For our business plan we considered building a smart jammer. When the signals that are present in the spectrum are identified a list of frequencies that need to be jammed can be generated. To do the actual jamming we decided to build a simple device to block one programmable frequency. The number of transmitters should scale with the number of expected signals in the frequency range that needs to be jammed.

In this section we will discuss the working of this jammer. We will begin with a block diagram of the transmitter, and we will then present the complete schematics and the PCB. Unfortunately, because this was a small side project, we did not have enough time to actually finish the design and build it. We designed a schematic and pcb footprints, simulated the essential parts. The we did not have enough time to design a noise source and do the routing and simulation of the PCB.

Block diagram

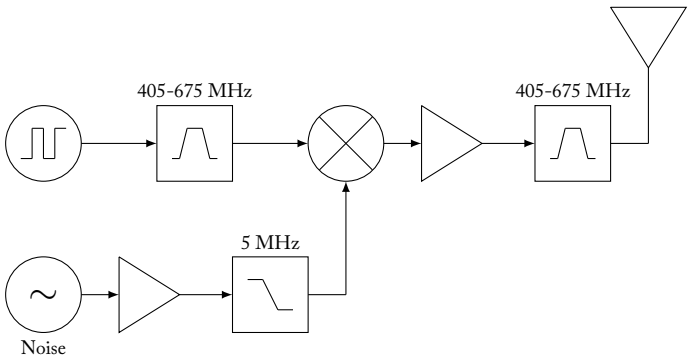


Figure G.1: Block diagram of the jammer

The block diagram of the jammer can be found in Figure G.1. The jammer works by generation a square wave with a programmable clock generator IC. These ICs are much cheaper than a solution with a very broadband voltage controlled oscillator and a PLL. We chose a specific IC with a frequency range of 3 MHz to 416 MHz. In

this example we want to jam the band including the 446 MHz frequency of license free radios. Our oscillator only operates up to a frequency of 416 MHz, therefore we need to select a high harmonic, in this case the third, and filter all of the other components of the square wave. In this case we use a bandpass filter that spans the 405 MHz to 675 MHz. The bandwidth is determined by the frequency of the fifth harmonic. At 405 MHz the base frequency is 135 MHz, and the fifth harmonic 675 MHz. We created a small script to generate a complete table of needed filters to span the whole range of 3 MHz to 3 GHz. These results can be found in Table G.1.

After the filtering the signal is mixed with a band-limited noise source, and then amplified and filtered again. This should produce a band limited signal with noise at the desired frequency.

Table G.1: Filter frequencies for frequency range of 3 MHz to 3179 MHz

Harmonic	Filter low [MHz]	Filer high [MHz]	LO low [MHz]	LO high [MHz]
1	3	9	3	9
1	9	27	9	27
1	27	81	27	81
1	81	243	81	243
3	243	405	81	135
3	405	675	135	225
3	675	1125	225	375
5	1125	1575	225	315
7	1575	2025	225	289
7	2025	2601	289	371
9	2601	3179	289	353

Schematics

The complete schematics can be found in Appendix H. Only the noise source is not implemented yet.

Filter design

For this implementation we chose a frequency range of 405 MHz to 675 MHz. We designed[1] a sixth order Chebyshev bandpass filter, with a ripple of 1.0 dB in the passband, a characteristic impedance of 50 Ω. We also designed a matching network to match the 50 Ω of the filter to the 180 Ω input impedance of the mixer. The schematics of the filter can be found in Figure G.2.

Microcontroller

A microcontroller was needed to control the clock generator IC and communicate with the PC. We decided to use an Atmega 328P. This is the same microcontroller used as on the very widespread Arduino boards. Therefore a lot of code is already

[1] Designed with `http://www-users.cs.york.ac.uk/~fisher/lcfilter/`

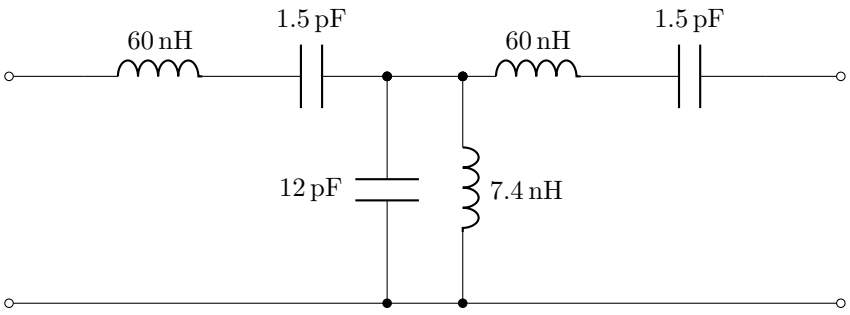


Figure G.2: The designed bandpass filter

available. This would allow us to develop the firmware in the limited time we had left.

The microcontroller connects to the PC with a special USB to Serial cable produced by FTDI. With this setup we can easily send commands from our python program to the microcontroller.

Simulations

To verify the working of the system we built a SPICE simulation of a simplified version of the schematic, see Figure G.3. The resistor of 1 MΩ was added to prevent a SPICE error about inductor loops. This includes the filter in Figure G.2 and a matching network to match to 50 Ω source to the 180 Ω impedance of the mixer.

Filters

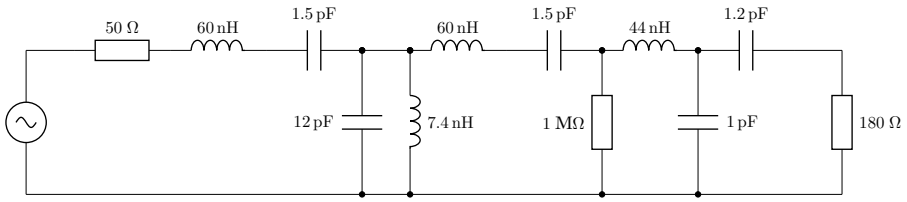


Figure G.3: Schematic of the filter and matching network used in the simulation

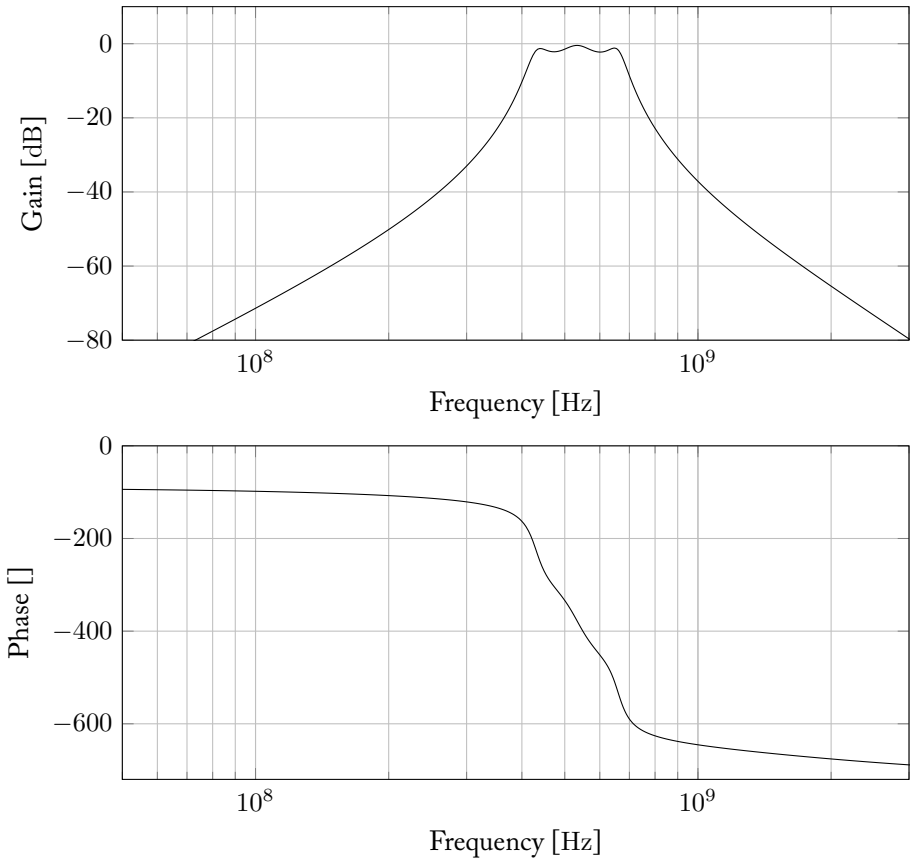


Figure G.4: Bode plot of the designed filter and matching network

If we look at the bode plot in Figure G.4 we can see the passband is in the frequency range we expected. The phase lag in the upper part of the filter is large, but expected for a sharp filter.

Complete System

We also did a simulation of the complete system, to check if our assumptions about the suppression of the unwanted harmonics worked correctly. We replaced the clock generator IC by a square wave source and the noise source by a theoretical noise source. The mixer was replaced by a theoretical multiplier. The resulting schematic can be found in Figure G.5.

The results of the simulation can be found in Figure G.6. We can see that the system works as we expected. The wanted third harmonic of the original signal is more than 20 dB stronger than the other signals. This is good enough for our purposes.

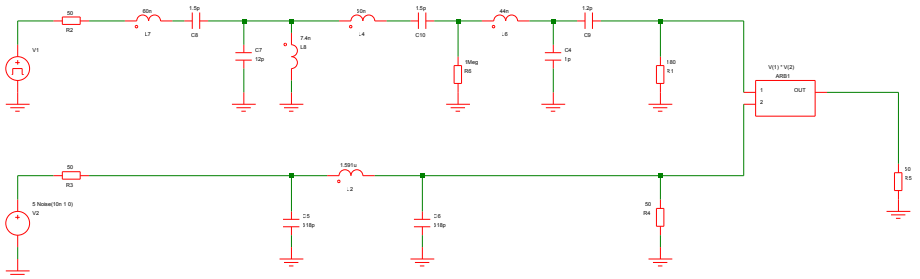


Figure G.5: The schematic used for the system simulation

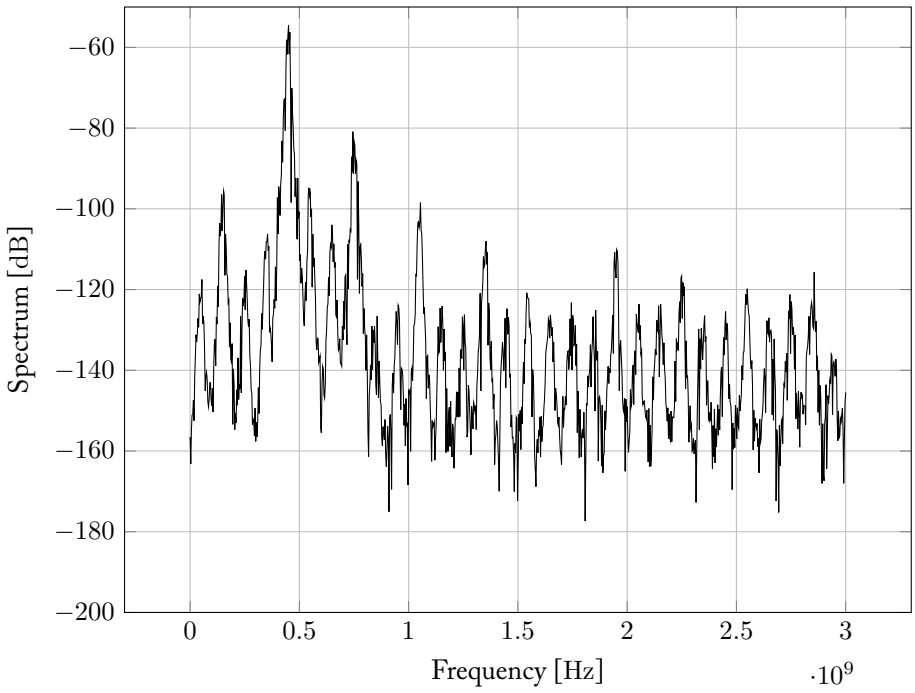


Figure G.6: Spectrum plot of the system simulation

PCB

During the schematic design we drew all the footprint and 3d-models of the part. Therefore the routing of the low frequency signals on the PCB should not be much work. The difficulty arises when routing high frequency wires where the physical properties of the PCB become a factor. Fortunately we used Altium to design our schematic and PCB we can make use of the signal integrity analysis. With this special tool we can calculate the influence of trace length on the signal integrity.

In Figure G.7 a first impression of the PCB can be seen. This is only mock-up, no actual routing or placement is done.

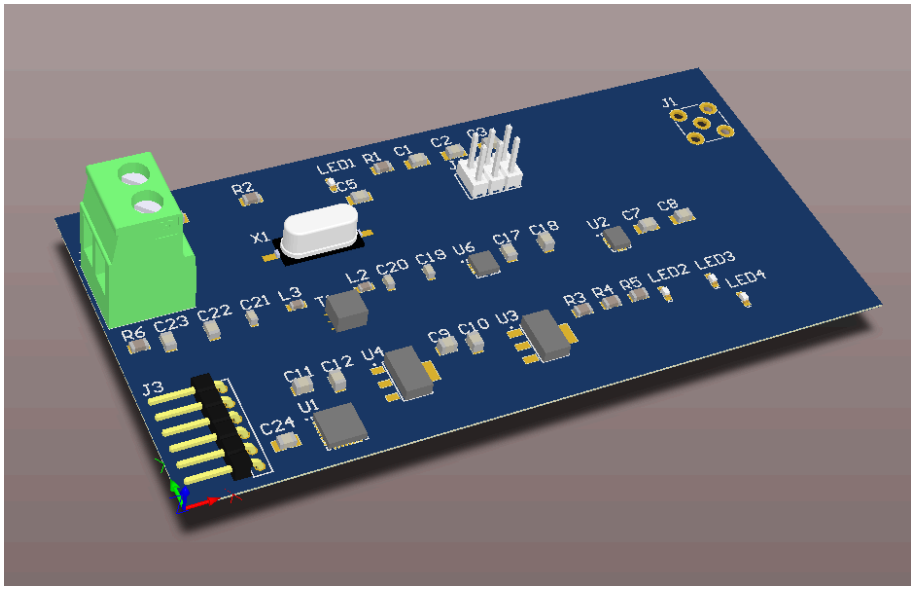
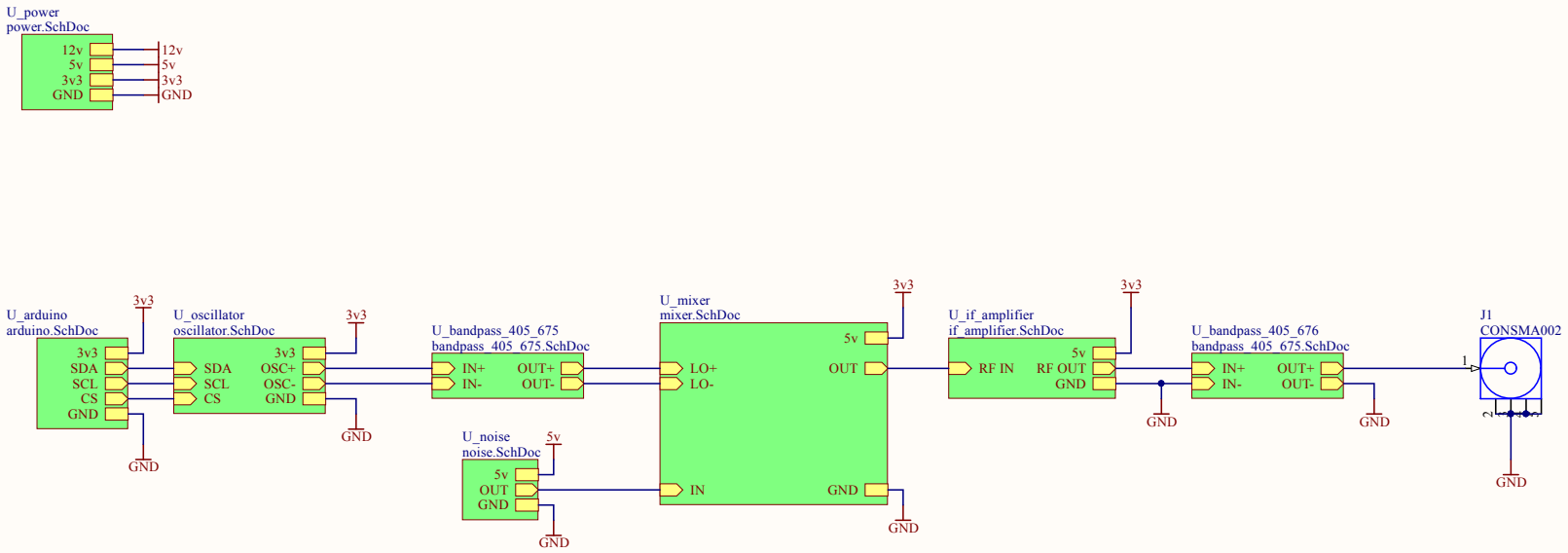
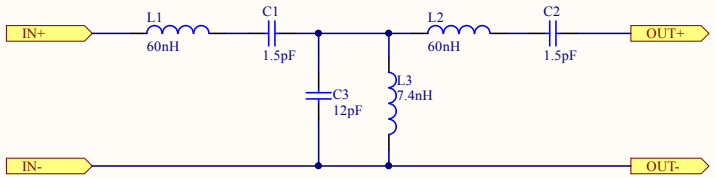


Figure G.7: 3D view of the designed PCB

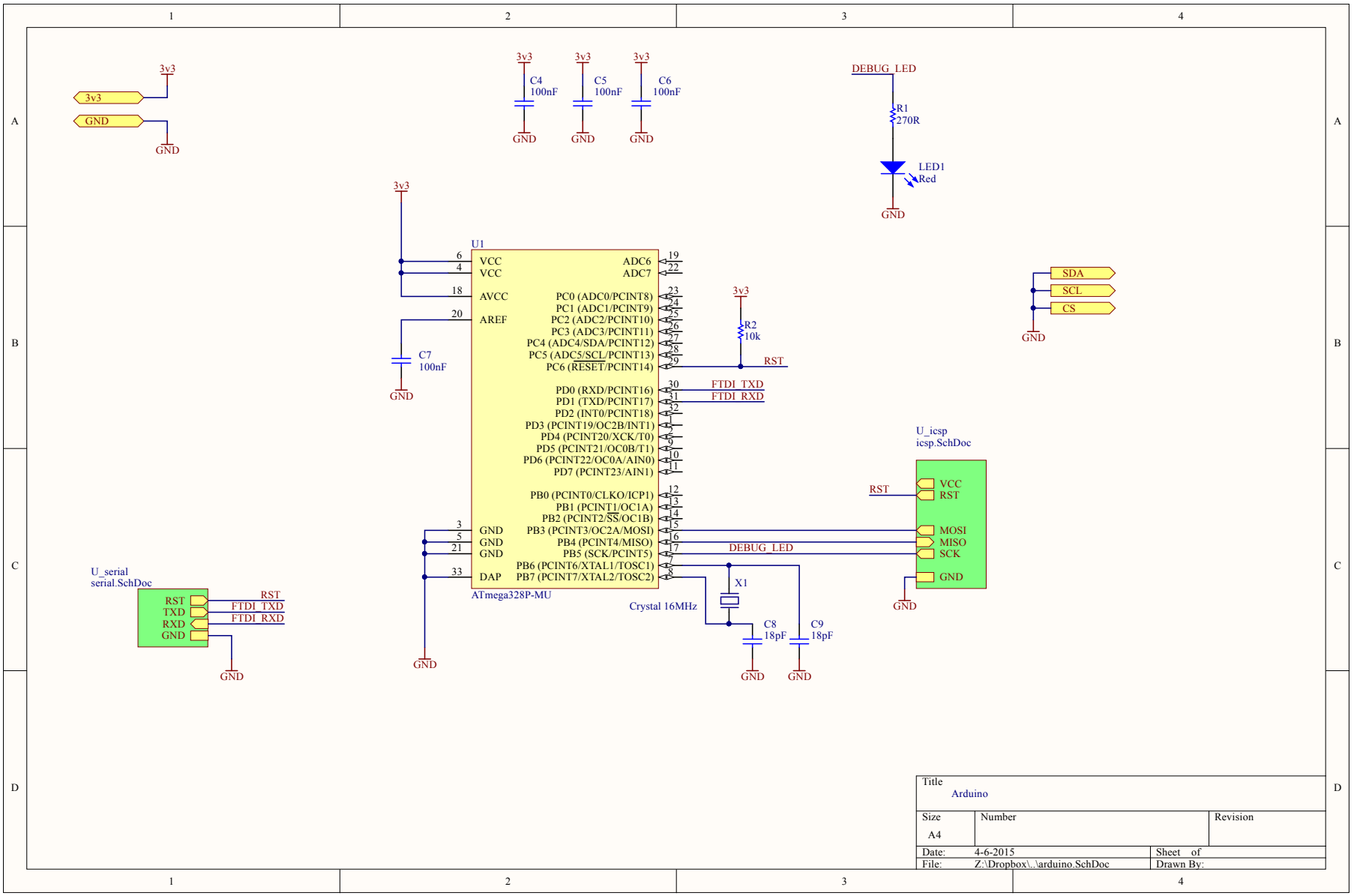
H | Schematics of the jammer

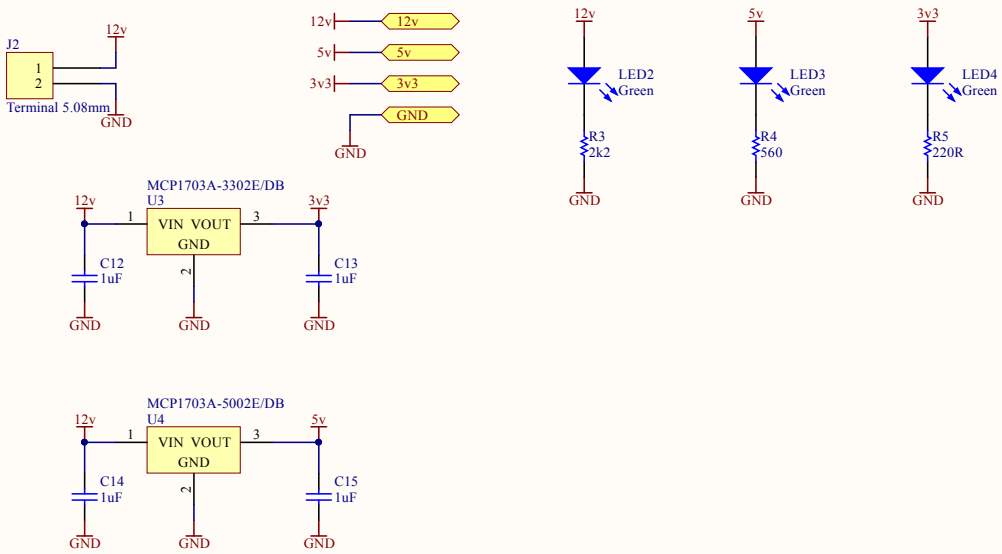


Title		
Size: A4	Number	Revision
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\main.SchDoc		Drawn By:

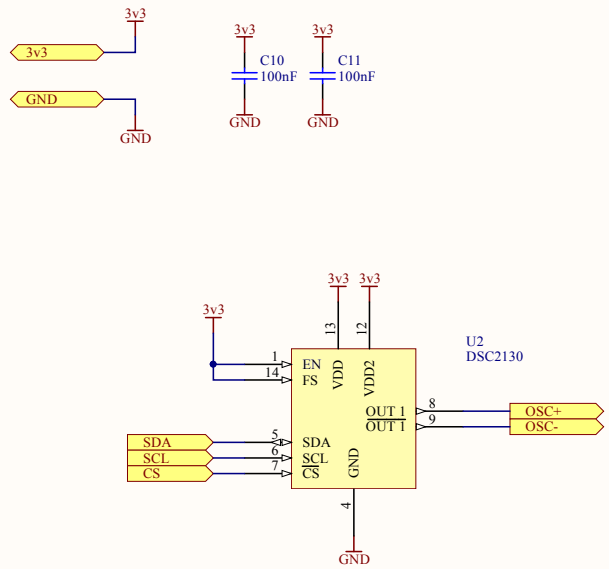


Title: Bandpass 405Mhz - 675Mhz, Zin = 50 Ohm		
Size: A4	Number:	Revision:
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\bandpass_405_675.SchDoc		Drawn By:

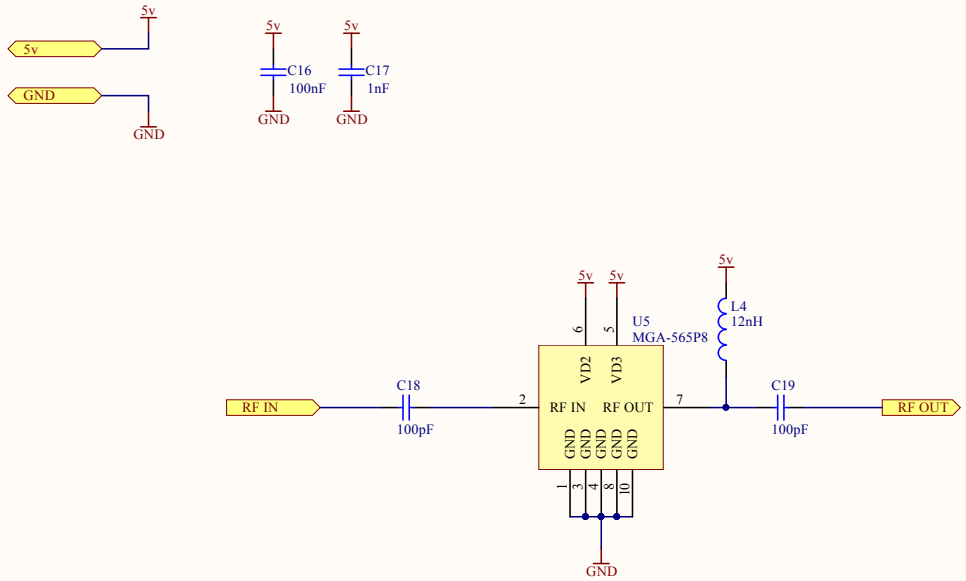

3v3
GND
C4
100nF
C5
100nF
C6
100nF
DEBUG_LED
R1
270R
LED1
Red
U1
VCC
VCC
AVCC
AREF
C7
100nF
R2
10k
RST
SDA
SCL
CS
PC0 (ADC0/PCINT8)
PC1 (ADC1/PCINT9)
PC2 (ADC2/PCINT10)
PC3 (ADC3/PCINT11)
PC4 (ADC4/SDA/PCINT12)
PC5 (ADC5/SCL/PCINT13)
PC6 (RESET/PCINT14)
PD0 (RXD/PCINT16)
PD1 (TXD/PCINT17)
PD2 (INT0/PCINT18)
PD3 (PCINT19/OC2B/INT1)
PD4 (PCINT20/XCK/T0)
PD5 (PCINT21/OC0B/T1)
PD6 (PCINT22/OC0A/AIN0)
PD7 (PCINT23/AIN1)
PB0 (PCINT0/CLKO/ICP1)
PB1 (PCINT1/OC1A)
PB2 (PCINT2/SS/OC1B)
PB3 (PCINT3/OC2A/MOSI)
PB4 (PCINT4/MISO)
PB5 (SCK/PCINT5)
PB6 (PCINT6/XTAL1/TOSC1)
PB7 (PCINT7/XTAL2/TOSC2)
ADC6
ADC7
GND
GND
GND
DAP
FTDI_TXD
FTDI_RXD
DEBUG_LED
ATmega328P-MU
X1
Crystal 16MHz
C8
18pF
C9
18pF
U_icsp
icsp.SchDoc
VCC
RST
MOSI
MISO
SCK
GND
U_serial
serial.SchDoc
RST
TXD
RXD
GND
Title
Arduino
Size
A4
Number
Revision
Date: 4-6-2015
File: Z:\Dropbox\..\arduino.SchDoc
Sheet of
Drawn By:



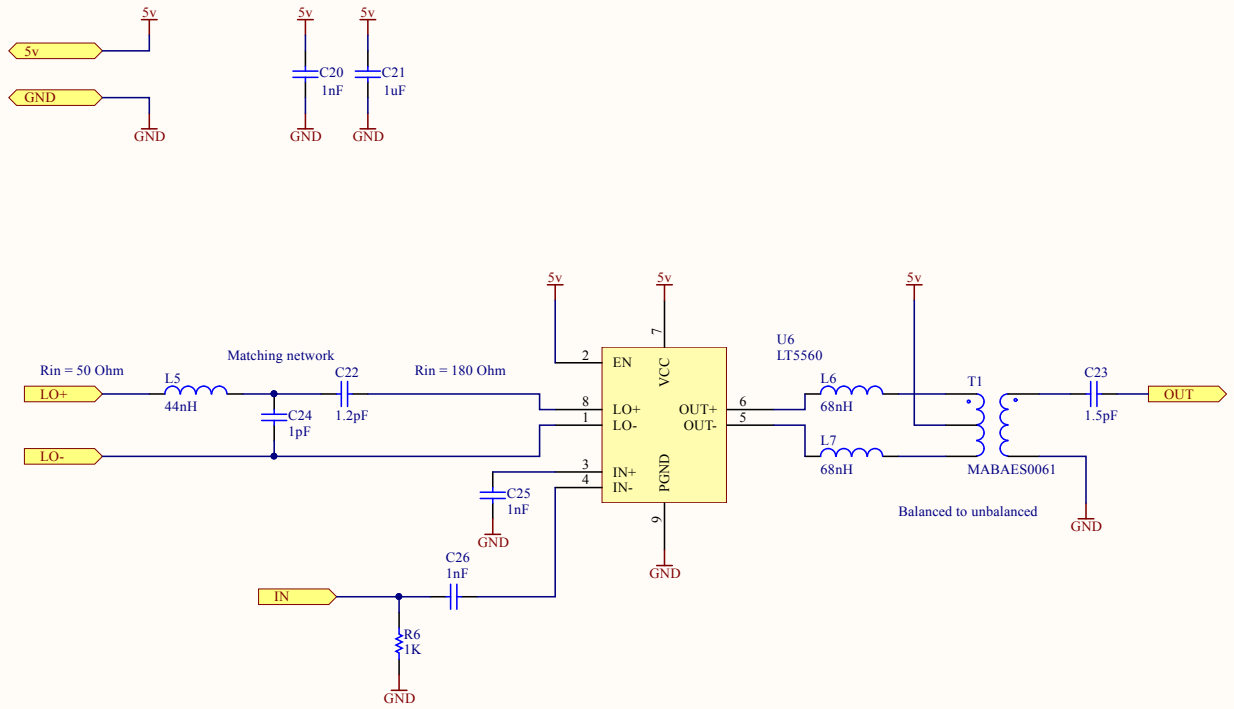
Title: Power management		
Size: A4	Number:	Revision:
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\power.SchDoc		Drawn By:



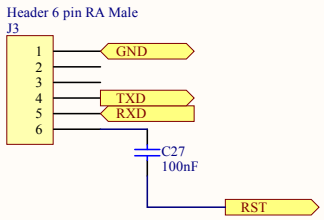
Title: Programmable Oscillator		
Size: A4	Number:	Revision:
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\oscillator.SchDoc		Drawn By:



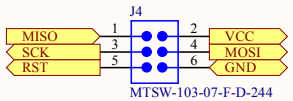
Title: IF Amplifier		
Size: A4	Number:	Revision:
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\if_amplifier.SchDoc		Drawn By:



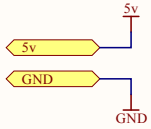
Title: Mixer		
Size: A4	Number:	Revision:
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\mixer.SchDoc		Drawn By:



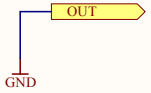
Title	Connection for FTDI Cable
Size	A4
Number	
Revision	
Date:	4-6-2015
File:	Z:\Dropbox\..\serial.SchDoc
Sheet	of
Drawn By:	



Title: In circuit programming cable connector		
Size: A4	Number:	Revision:
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\icsp.SchDoc		Drawn By:



Not Yet Implemented



Title: Noise source, band limited to 10MHz		
Size: A4	Number:	Revision:
Date: 4-6-2015		Sheet of
File: Z:\Dropbox\..\noise.SchDoc		Drawn By: