CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING

EURECOM

AUSTRIAN INSTITUTE OF TECHNOLOGY

FREQUENCY-HOPPING PARAMETER ESTIMATION FOR UNMANNED AERIAL VEHICLE RADIO LINKS

MASTER THESIS BY **JIŘÍ SKOŘEPA**

MASTER PROGRAMME AT CTU:
ELECTRONICS AND COMMUNICATIONS

BRANCH OF STUDY:
RADIO AND OPTICS

MASTER PROGRAMME AT EURECOM:
MOBILE COMPUTING SYSTEMS

SUPERVISORS:
DOC. DR. ING PAVEL KOVÁŘ FROM CTU
DR.TECHN. FLORIAN KALTENBERGER FROM EURECOM
PRIV.-DOZ. DI DR.TECHN. THOMAS ZEMEN FROM AIT

VIENNA, MARCH 2019

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ

EURECOM

AUSTRIAN INSTITUTE OF TECHNOLOGY

ODHAD PARAMETRŮ FHSS RADIOVÝCH LINEK PRO BEZPILOTNÍ PROSTŘEDKY UAV

DIPLOMOVÁ PRÁCE: **JIŘÍ SKOŘEPA**

STUDIJNÍ PROGRAM NA ČVUT:
ELEKTRONIKA A KOMUNIKACE

STUDIJNÍ OBOR:
RÁDIOVÁ A OPTICKÁ TECHNIKA

STUDIJNÍ PROGRAM EURECOM:
MOBILE COMPUTING SYSTEMS

VEDOUCÍ PRÁCE:
DOC. DR. ING PAVEL KOVÁŘ FROM CTU
DR.TECHN. FLORIAN KALTENBERGER FROM EURECOM
PRIV.-DOZ. DI DR.TECHN. THOMAS ZEMEN FROM AIT

VÍDEŇ, BŘEZEN 2019

CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING

EURECOM

AUSTRIAN INSTITUTE OF TECHNOLOGY

L'ESTIMATION DES PARAMÈTRES DE L'ÉTALEMENT DE SPECTRE PAR SAUT DE FRÉQUENCE LIAISONS RADIO DE VÉHICULES AÉRIENS SANS PILOT

THÈSE DE MASTER PAR **JIŘÍ SKOŘEPA**

PARCOURS DE MASTER À CTU:
ELECTRONIQUE ET COMMUNICATION

SPÉCIALITÉ:
RADIO ET OPTIQUE

PARCOURS DE MASTER À EURECOM:
SYSTÈMES POUR L'INFORMATIQUE MOBILE

SUPERVISEURS:
DOC. DR. ING PAVEL KOVÁŘ À CTU
DR.TECHN. FLORIAN KALTENBERGER À EURECOM
PRIV.-DOZ. DI DR.TECHN. THOMAS ZEMEN À AIT

VIENNE, MARS 2019


EURECOM
Sophia Antipolis


AUSTRIAN INSTITUTE
OF TECHNOLOGY

Abstract

Recently, there has been an expansion of unmanned aerial vehicles (UAV) – flying drones – in use. In this project, we focus on drones controlled by user's remote radio controller using a frequency-hopping spread spectrum (FHSS) modulation. The main goal is to estimate parameters of such modulated signal transmitted by a drone's controller. To accomplish the task, we are using a state-of-the-art software-defined radio (SDR) receiver. The first step is to introduce and compare several time-frequency analysis tools, specifically the Short-time Fourier transform (STFT), the wavelet transform (WT) and the Wigner-Ville distributions (WVD). Deploying properly configured SDR receiver, a record of a realistic unknown drone controller's FHSS signal is taken. After time-frequency analysis of the recorded signal, we introduce an adaptive threshold based on a constant false alarm ratio (CFAR) algorithm to estimate the parameters of the recorded FHSS signal. At the end of our work, we implement an energy detector based on the STFT together with the selected CFAR algorithm into an FPGA to detect the UAV's FHSS signal in real-time.

Keywords

unmanned aerial vehicle (UAV), frequency-hopping spread spectrum (FHSS), software-defined radio (SDR), short-time Fourier transform (STFT), wavelet transform (WT), Wigner-Ville distribution (WVD), pseudo Wigner-Ville distribution (PWVD), smoothed Wigner-Ville distribution (SPWVD), energy detection, Constant False Alarm Ratio (CFAR), FPGA

Abstrakt

V posledních letech došlo k masovému rozšíření drobných bezpliotních letounů (UAV), tzv. dronů. V této práci se zaměříme na drony ovládané uživatelem na dálku pomocí rádiového signálu využívajícího metodu rozprostřeného spektra frekvenčním skákáním (FHSS). Cílem práce je odhadnout parametry takto modulovaného signálu vyslaneho příslušným ovladačem dronu. Ke splnění tohoto cíle bude využito nejmodernějšího softwarově definovaného rádiového přijímače.Prvním krokem je představení a srovnání nástrojů pro časově-frekvenční analýzu, konkrétně krátkodobou Fourierovu transformaci, vlnkovou transformaci a Wigner-Villovu distribuci (WVD). S použitím patřičně nakonfigurovaného softwarově definovaného radiového přijímače provedeme měření a záznam skutečného signálu vyslaného ovladačem dronu. Po časově-frekvenční analýze měřeného signálu je představen adaptivní práh s použitím Constant False Alarm Rate (CFAR) algoritmu pro odhad parametrů FHSS modulace. Závěr práce je věnován implementaci energetického detektoru na bázi STFT a vybraného CFAR algoritmu do FPGA pro detekování FHSS signálu v reálném čase.

Klíčová slova

unmanned aerial vehicle (UAV), spektrum rozprostřené frekvenčním skákáním FHSS), softwarově definované rádio (SDR), krátkodobá Fourierova transformace (STFT), vlnková transformace (WT), Wigner-Villova distribuce (WVD), pseudo Wigner-Villova distribuce (PWVD), vyhlazená pseudo Wigner-Villova distribuce (SPWVD), energetický detektor, Constatnt False Alarm Ratio (CFAR), FPGA

Résumé

Récemment, l'utilisation de véhicules aériens sans pilote (UAV) a augmenté. Dans ce projet, nous nous concentrons sur les drones contrôlés par le contrôleur radio utilisant un spectre étalé à sauts de fréquence (FHSS). L'objectif principal est d'estimer les paramètres d'un tel signal modulé transmis par le contrôleur d'un drone. Pour accomplir cette tâche, nous utilisons un récepteur radio logiciel (SDR) de dernière génération. La première étape consiste à introduire et à comparer plusieurs outils d'analyse temps-fréquence, en particulier la transformée de Fourier à court terme (STFT), la transformée en ondelettes (WT) et la distribution de Wigner-Ville (WVD). En déployant un récepteur radio correctement configuré, un enregistrement du signal d'un contrôleur de drone inconnu réaliste est pris. Après analyse temps-fréquence du signal enregistré, nous introduisons un seuil adaptatif basé sur un algorithme de taux de fausse alarme constant (CFAR) pour estimer les paramètres du signal enregistré. À la fin de nos travaux, nous mettons en œuvre un détecteur d'énergie basé sur STFT ainsi qu'un algorithme CFAR sélectionné dans le FPGA pour détecter le signal en temps réel.

Mots-Clés

véhicules aériens sans pilote (UAV), spectre étalé à sauts de fréquence (FHSS), récepteur radio ogiciel (SDR), transformée de Fourier à court terme (STFT), transformée en ondelettes (WT), distribution de Wigner-Ville (WVD), détecteur d'énergie, algorithme de taux de fausse alarme constant (CFAR), FPGA

Acknowledgement

It is my pleasant responsibility to sincerely thank to my supervisors Pavel Kovář, Florian Kaltenberger and Thomas Zemen for their invaluable advice and guidance. I would also like to express my thanks to David Löschenbrand and Markus Hofer for their time invested to productive discussions, as well as to other colleagues from the department of Security & Communication Technologies at Austrian Institute of Technology. I highly appreciate the given opportunity to work with all the colleagues there and with the equipment provided by AIT.

I am especially grateful to Zdeněk Bečvář from Czech Technical University in Prague who encouraged me in studying abroad and helped me to arrange the scholarship support. Last but certainly not least, I am thankful to my family and friends for their unceasing support.

Declaration of Originality

I hereby declare that this master thesis is the result of my own independent work. I affirm that, to my best of consciousness and conscience, all the used literature sources, ideas, quotations, or any other material of other people included in my thesis are correctly referenced in compliance with the standard referencing practices.

Author

In Vienna, March 3th 2019

Official Description

Design a software-defined-radio system that is able to identify the unknown parameter of an UAV radio communication link. Literature research on principal methods for frequency-hopping parameter estimation. Measure unknown UAV frequency hopping spread spectrum (FHSS) modulated downlink signal with a state-of-the-art software-defined radio testbed. Implementation of a suitable parameter estimation method in MATLAB/Python/LabVIEW. Validation of the implementation in a real-world scenario.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Skořepa** Jméno: **Jiří** Osobní číslo: **425045**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra elektromagnetického pole**

Studijní program: **Elektronika a komunikace**

Studijní obor: **Rádiová a optická technika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Odhad parametrů FHSS radiových linek pro bezpilotní prostředky UAV

Název diplomové práce anglicky:

Frequency-Hopping parameter estimation for unmanned aerial vehicle (UAV) radio links

Pokyny pro vypracování:

Navrhněte radiový systém založený na softwarovém rádiu (SDR) pro identifikaci neznámých parametrů radiové linky bezpilotních prostředků UAV. Proveďte průzkum literatury ohledně metod odhadu parametrů systémů s kmitočtovým skákáním. Změřte neznámou komunikaci s UAV s kmitočtovým skákáním a proveďte odhad parametrů současným softwarem pro SDR přijímače. Napište vlastní software pro odhad parametrů v Matlabu, Pythonu nebo LabView. Software ověřte pomocí reálných signálů.

Seznam doporučené literatury:

[1] Rohde, U,. Whitaker, J., Zahnd, H.: Communications Receivers: Principles and Design, Fourth Edition, McGraw-Hill 2017, ISBN 10: 0071843337
[2] Torrieri, D.: Principles of Spread-Spectrum Communication Systems, First Edition, Springer 2005, ISBN: 978-1-4419-3558-8
[3] Lyons, R. G.: Understanding Digital Signal Processing, Third Edition, Prentice Hall 2011, ISBN-10: 0137027419
[4] Harris, F. J.: Multirate Signal Processing for Communication Systems, Prentice Hall 2004, ISBN: 0131465112
[5] Ling, F., Proakis, J.: Synchronization in Digital Communication Systems, Cambridge University Press 2017, ISBN: 9781316335444, doi:10.1017/9781316335444

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Dr. Ing. Pavel Kovář, katedra radioelektroniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **09.10.2018** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

doc. Dr. Ing. Pavel Kovář podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Abbreviation List

Abbreviation	Meaning
5G	5th generation mobile communication networks
ADC	Analog-to-digital converter
AWGN	Additive white Gaussian noise
GSM	Global System for Mobile communications
GPS	Global Positioning System
BT	Bluetooth
BRAM	Block random access memory
CLB	Configurable logic block
CPU	Central processing unit
CFAR	Constant false alarm ratio
CA-CFAR	Cell Average-CFAR
CAGO-CFAR	Cell Average Greatest Of-CFAR
CASO-CFAR	Cell Average Smallest Of-CFAR
CORDIC	Coordinate Rotation Digital Computer
OS-CFAR	Order Statistics-CFAR
DAC	Digital-to-analog converter
DC	Direct current
DFT	Discrete Fourier transform
DJI	SZ DJI Technology Co., Ltd.
DSP	Digital signal processor/ing
DDC	Digital down-converter
DDS	Direct digital synthesis
DVB-T	Digital Video Broadcasting — Terrestrial
DSLR	Digital single-lens reflex camera
DSSS	Direct-sequence spread spectrum
ESPRIT	Estimation of Signal Parameters via Rotational Invariance Techniques
FPGA	Field-programmable gate array
FT	Fourier transform
FFT	Fast Fourier transform
FHSS	Frequency-hopping spread spectrum
FIFO	First in, first out
FSK	Frequency shift keying

Abbreviation	Meaning
I16	Integer 16 bit
IF	Intermediate frequency
IP	Intellectual property
LO	Local oscillator
LORA	Long Range
LTE	Long-Term Evolution
LUT	Look-up table
MATLAB	Matrix Laboratory
MUSIC	Multiple Signal Classification
NI	National Instruments
OFDM	Orthogonal frequency-division multiplexing
PC	Personal computer
PCIe	Peripheral Component Interconnect Express
PLL	Phase-locked loop
PWVD	Pseudo Wigner-Ville distribution
RADAR	Radio detection and ranging
RAM	Random access memory
RC	Remote controller
RF	Radio-frequency
ROC	Receiver operating characteristics
SPA	Spectrum analyzer
SDR	Software-defined radio
SPWVD	Smoothed pseudo Wigner-Ville distribution
SNR	Signal-to-noise ratio
STFT	Short-time Fourier transform
TFA	Time-frequency analysis
U32	Unsigned integer 32 bit
UAV	Unmanned aerial vehicle
USB	Universal serial bus
USRP	Universal software radio peripheral
VCO	Voltage controlled oscillator
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WIFI	Wireless Fidelity
WVD	Wigner-Ville distribution
WT	Wavelet transform

Contents

Chapter 1

Introduction

Having read the title of this thesis might leave many readers with a feeling of confusion what is being dealt with inside this document. Instead of saying "unmanned aerial vehicle (UAV)" we could also say remote-controlled drones, which have recently reached the level where they are affordable as relatively inexpensive commercial products and easily manageable for users (a large part of the society). The market offers a wide variety of those pilotless flying machines matching the desired application [1], which children can play with or race with [2]; [3] for recreational aerial photography or [4], [5] as a tool for cinematographers helping them in production of a new feature film; [6], [7] incorporated in public safety services; [8] as a farmer or postman [9] in a close future. With incoming 5G communication networks, we can also expect deployment of drones in the mobile network infrastructure [10].

The typical connection between the remote controller and the drone is a wireless link based on radio frequency electromagnetic waves allowing transmission of commands – data – to the drone[1] The communication in the direction from a UAV to a controller is also possible for streaming the video footage recorded by a UAV to a controller device. To complete the thesis heading explanation, within this document, we will be dealing solely with radio links providing communication from a remote-controller device to the drone. We will not be interested in the exact data being sent between those devices. We intend to identify how this radio communication is performed. Since there are plenty of radio communication alternatives[2], we will focus merely on one of them, i.e., the Frequency-Hopping Spread Spectrum technique (FHSS). Our work will also be restricted on the commercially sold drones only.

[1] The word *drone* is a causal expression for an unmanned aerial vehicle (UAV). In a daily life, *drone* is more common title than the UAV.

[2] Direct-Sequence Spread Spectrum (DSSS), Orthogonal Frequency Division Multiplexing (OFDM), etc.

Even drones may look inconspicuous, with their recently increased availability the potential problems connected to safety and security have emerged[3]. We already understood that they could carry several kilograms of load, no matter if it is a camera, package being delivered to a customer, or anything else, hence the drones can be purposefully misused for illegal activity. The potential threat can be even more unambiguous if we imagine a scenario where a child unintentionally flies a drone over the neighbor's backyard or a restricted area. By the landowner, this action could be perceived as undesirable or an act of infringement. Thus an option which would help to resolve such inconvenient situations is being sought.

One choice could be to forbid all UAVs for good. Nevertheless, then we would lose all the benefits this technology offers. The reasonable compromise between the full prohibition and irresponsible neglection of the risks is proper regulation [12]. At this moment, a challenge comes for engineers since each regulation requires a tool for monitoring and control [13].

Deployed Drone Detection Systems

Presently, many companies are dealing with drone monitoring. Some of the companies built a dominant business [14], others rely on the developed technology they already have [15], [16]. Most of the companies provide a system solution. That means that several areas of drone detection are deployed. The drone cruising the sky interacts with its surrounding environment in several domains. First, it is not invisible, therefore vision detection can be applied. The mechanical motion of the engine parts and blades produce audible sound, thus a sound detection can be used. Also, it takes a volume in space, though very small, it can be detected by a radar [17], [18]. For us, the most appealing characteristic is its requirement of the reception of a radio signal coming from the remote control. Thanks to the nature of electromagnetic wave propagation, we can receive that signal at our receiver even if it was not intended for our receiver.

Thesis Concept

The objective of this work is to contribute to the security subject of unmanned aerial vehicles within the field of radio frequency signal detection. We are aiming at detection of a drone's signal and the estimation of the signal's behavior as time passes by. We will be solely interested in the instantaneous signal frequency – determine the radio channel where the signal is being transmitted. One of many specifications of an FHSS link is a regular or irregular change of the radio frequency channel. As we will explain later in the text, the communication link between the remote controller and the drone is not being established at only one frequency channel all the time. Furthermore, we will be curious how long the signal persists at that specific frequency. Knowing that information, we can attempt to disrupt the connection between the controller and the drone by transmitting a signal at the precisely same frequency channel where UAV's communication is operating. The engineering terminology calls this technique

[3] A recent incident caused a spur at Gatwick airport [11].

jamming, and it is the next step for a UAV security system. Thus, this thesis provides the initial step for such a system when the drone's signal needs to be detected first.

Organistaion of The Thesis

Chapter two describes the problem of FHSS radio communication links, their purpose and characteristics. The third chapter introduces the concept of a software-defined radio, which is the equipment we use in this project. We consider as necessary to state its capabilities and weaknesses. In the fourth chapter, we prepared a simulation of a FHSS radio signal. This simulation is used to compare several time-frequency analysis techniques – potential candidates solving our task. Chapter five focuses thoroughly on the time-frequency analysis technique we decided to use in our project.

The sixth chapter documents the signal measurements we recorded with the software-defined receiver. Meaningful results can be found in this chapter. Chapter seven follows the results from chapter five. It provides a mechanism how to simplify and automate the process of FHSS signal detection. Within the chapter eight, we use all the information and knowledge from the previous chapters to develop a real-time application suitable for our software-defined radio equipment. The conclusion and a short discussion about the problems we faced in our project and which of them we accomplished to solve is mentioned in the final chapter nine. Additional content is also placed in the appendix, mainly the source codes.

Chapter 2

The UAV Radio Communication Link

Within this chapter, we describe the wireless communication scenario between the remote controller and the drone, which is being controlled. We explain what the typical characteristics of such a link are, and why and how they need to be taken into our consideration. Since our task is to deal with an unknown UAV's radio link, we will refer to corresponding literature or well-known standardized techniques, which use similar concepts as UAV does. Describing the UAV communication system, we use an analogy to clarify the related issues. Knowing basic principles of digital modulation and spread spectrum methods is necessary for better understanding how our unknown radio communication between the drone and RC controller is established. This section intends to present useful and valid information, which justify the assumptions we will make later in our work.

2.1 Scenario Description

We can imagine a situation when a user of a drone manually sends position commands or instructions for equipment on the drone's board via the remote controller held in user's hands.

The expected UAV's communication system is depicted in the figure 2.2. In this project we are mainly interested in the parameters of the *modulator* block, where the FHSS signal is generated. We devote a special section to introduce the FHSS modulation process. Nevertheless, we need to remind, what creates and influence the UAV signal radio link so that we can make necessary assumptions since our FHSS signal is completely unknown.
Imagining the flying drone as illustrated in the figure 2.1, where the drone is controlled from the user's remote controlling device on the ground and what are the difficulties. The main challenge related to the remote controlling a flying machine is reliability of a radio communication link since nobody wants to lose control over the drone. The second one is latency. When the user commands the drone to turn left, the behavior of the drone should be immediate.

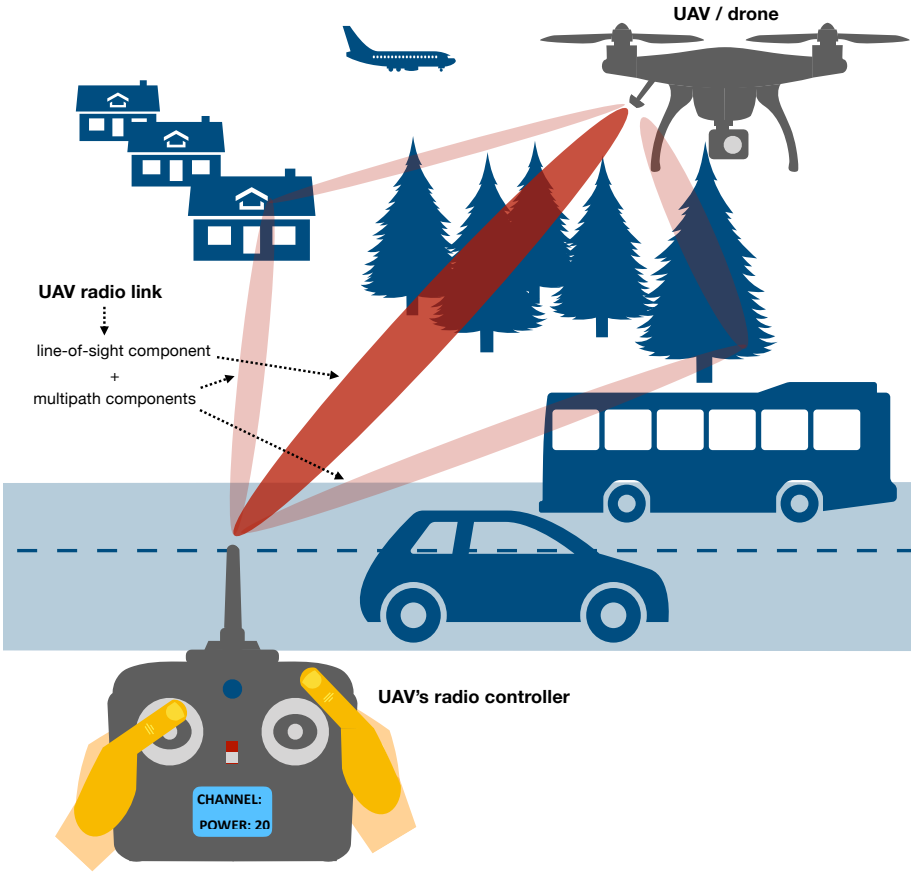


Figure 2.1 – An UAV (drone) flying in a realistic environment, controlled by a remote radio controller.

2.2 Frequency-hopping Spread Spectrum

We know that in the radio communication link, the data are modulated on the carrier with a certain frequency and radiated by the antenna of a transmitter in the form of electromagnetic energy to the environment, represented as a channel.

The signal is hopping from one frequency channel to another during the time. You can also understand the situation like the hopping spreads your channel among many other frequency channels within a new wider band since your communication is performed on that one channel only but in general, the channel can be selected from many other channels. There comes the expression *Frequency Hopping Spread Spectrum*.

Looking at the picture 2.3 above, the frequency-hopping pattern, which defines at which frequency channel and at which time the devices are communicating, is shown. This pattern can be arbitrary. It can be periodical, regular or random. Clearly, the transmitter and receiver need to know this pseudo-random pattern.

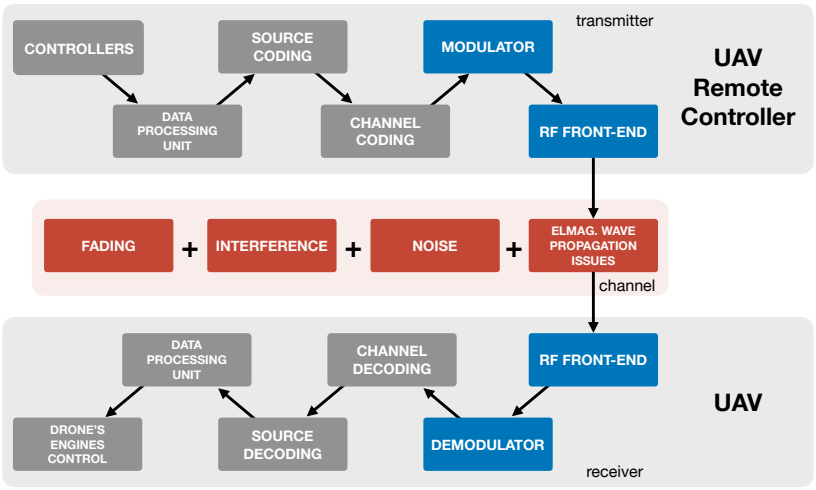


Figure 2.2 – Drone communication system – from pressing the button to the engines' action.

Moreover, the transmitter's and receiver's pseudo-random sequence must be synchronized in time and in frequency otherwise the data would be sent at a frequency where the receiver is not listening. Not every time this is an easy task. The price we usually pay for proper synchronization is time – latency.

We introduce a terminology which is typical for the FHSS concept description. Most of the expressions are taken from [19]. The rest of them was found in other books or discussions.

Center frequency: The frequency in the middle of the frequency channel. Typically identical with carrier frequency

Hopset: The set of M possible carrier frequencies $(f_1, f_2, ..., f_M)$ [19]

Hopping band: Frequency band over which the hopping occurs and which includes the M frequency channels [19].

Hoprate: The rate at which the carrier frequency changes [19]

Hop duration: The amount of time which is spent at one frequency channel.

Gap window: The time between the hops since it requires some time to tune from one carrier frequency to another and also to relax conditions for synchronization

All of those parameters, which define the FHSS signal, are depicted in the figure above 2.3. Hoprate, hop duration and hopset are the parameters which we are interested in within this project. Those we will try to estimate without a priori information about the radio link.

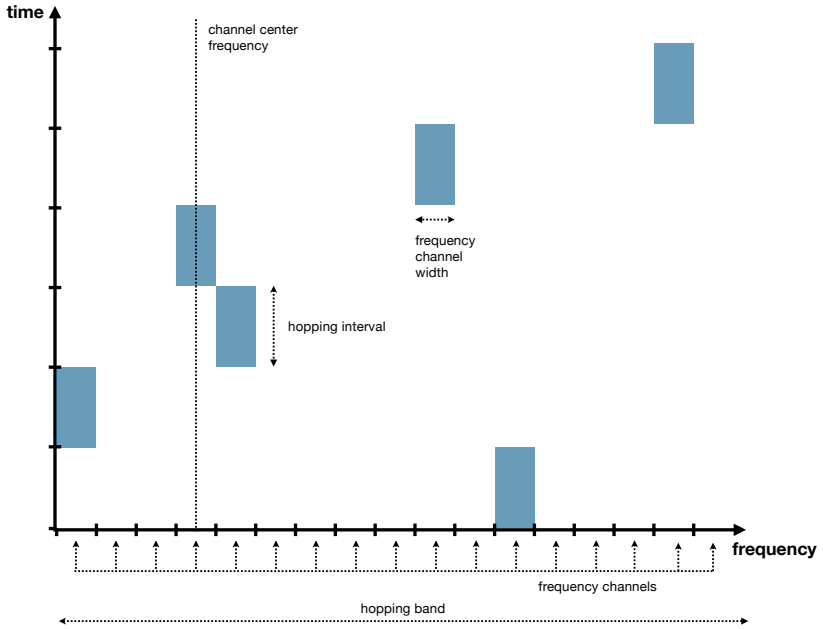


Figure 2.3 – Time-frequency chart of the frequency-hopping spread spectrum signal. Note that the pattern can be random in general.

2.2.1 Reasons for Frequency-hopping Spread Spectrum

Fading mitigation There are several reasons why is beneficial to deploy the FHSS. One of the problem is the frequency selective impulse response of the wireless channel caused by multiple propagation paths with different delay. Assuming that not all the frequency channels are bad, we can hop among all the frequencies and try to use the good channels and communicate at these. This short communicating interval lasting one hop duration can improve our chance of successful transfer of command data to the drone. We do not know which frequency channels are bad or good. Thus one possible scenario is merely to randomly[1] hop within the hopping band. Telling the truth, this is what most of the FHSS communication links do.

Interference Mitigation Since deploying an FHSS signal increases the width of the total band, we need to realize that the FHSS signal is not operating alone in the frequency spectrum band. There might be other drone pilots occupying arbitrary frequency channels within the same hopping band. Also, other wireless communication technologies providing a different kind of communication, for example, communication between sensors[2] may use the same band as your FHSS UAV radio communication link. Last but not least, Bluetooth and WIFI technologies may occupy some of the FHSS frequency channels also[3].

[1] hopping according to predefined pseudo-random sequence

[2] Zigbee, LORA, SIGFOX, ...

[3] Each of the technology typically operates within a certain ISM or SRD band, which are unlicensed bands for those purposes. We will talk about them later in the text.

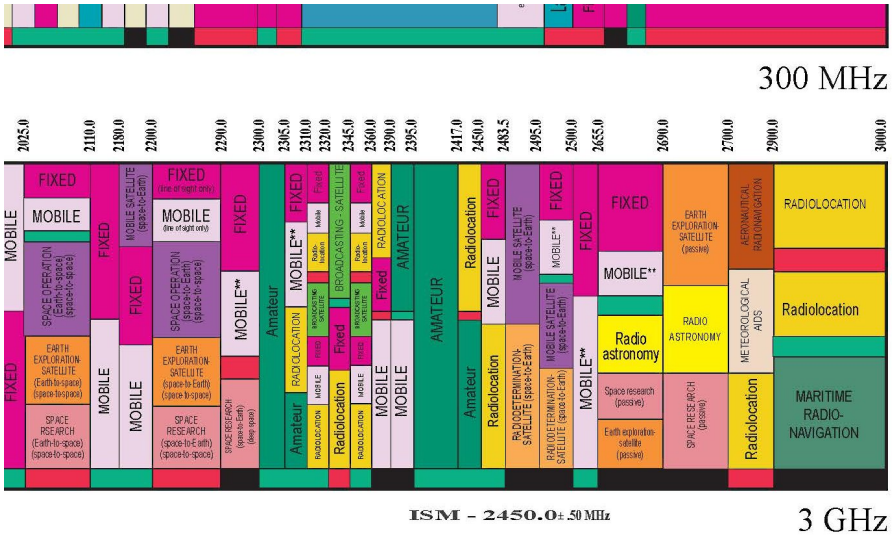


Figure 2.4 – Allocation of the 2.45 GHz ISM band reserved for "amateur" purposes. Keep in mind that in this band, WIFI, Bluetooth, Zigbee and many other technologies are operating thus mutually interfering. Taken from [20].

The reason why other technologies may use the FHSS signal's hopping band is that radio frequency spectrum – the general band from several kHz up to tens of GHz – is very precious to us. It is a kind of a natural resource. We cannot produce more spectra. Looking at the figure 2.4, we notice that there are portions of a spectrum – bands – which are called "Industrial, Scientific and Medicine band" (ISM). Those band can be used by anybody, who does not violate the regulations related to utilization. We dedicate a section in the text to explain details. For now, we can disclose that in most cases, the ISM bands are used for the FHSS drone communication.

Security A bright reader may point out that with the Frequency-Hopping Spread Spectrum signal, we also gain a security advantage. The transmitter and receiver must know the pseudo-random sequence according to which the signal is spread through the spectrum hopping band. If there were a violator, who would like to receive the FHSS communication, he would need to know the pseudo-random sequence. Also, he would need to synchronize the receiver with the FHSS signal.

2.3 FHSS System:

We now clarify the functionality of an UAV's communication system in detail. We surely remember the figure 2.2. For the UAV system, we need to modify the structure a bit. Inside the transmitter, after the modulator block, the modulated signal is mixed with another signal. The frequency of that signal corresponds to the pseudo-random sequence generated by the transmitter's controlling unit. This mixing process modulates the previously modulated signal. The result is the FHSS signal that carriers useful data and which frequency follows the pseudo-random pattern. Such waveform is forwarded to the RF front-end and radiated by the antenna.

$$s(t) = \sqrt{\frac{2E_b}{T_b}} \cos[2\pi f_j t + \phi(t) + \phi_i], (i-1)T_h \leq t \leq iT_h \tag{2.1}$$

where $\frac{E_b}{T_b}$ is the ratio of energy per bit and bit period, f_j is the channel carrier frequency from the set of possible hopping channels, i is the number of hop, ϕ_i is the random phase for the ith hop, T_h is the hop duration. Taken from [19] as formula 3.1.

The signal can be also directly spread in frequency even with only one modulating process, concurrently with the data signal modulation. The first mentioned design is depicted in 2.5. Each of the designs have its advantages and disadvantages. We will not investigate them within this thesis, but we must mention that this option is not possible always.

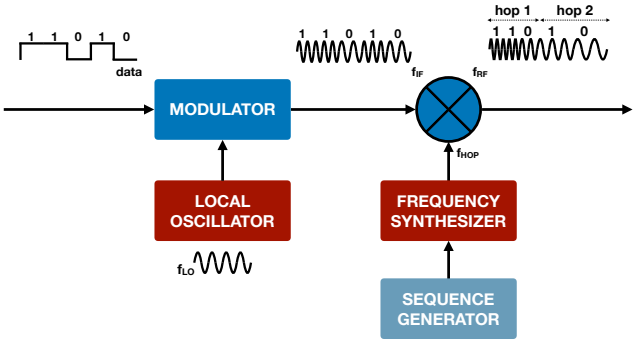


Figure 2.5 – FHSS modulator mixing the already modulated data on the hopped frequency according to the generated sequence.

Knowing that the receiver must keep the replica of the pseudo-random sequence stored, and that the receiver must be synchronized to the transmitted signal, we do not find anything confusing in the scheme below in 2.6.
By mixing down, which is an inverse process to the one in the transmitter, we obtain a signal which is not changing its frequency - hopping - anymore. Such process is called dehopping, and its output is signal with the modulated useful data.

Pseudo-random Sequence The pseudo-random sequence is a binary sequence, which determines the hopping frequency and the hop duration. It is

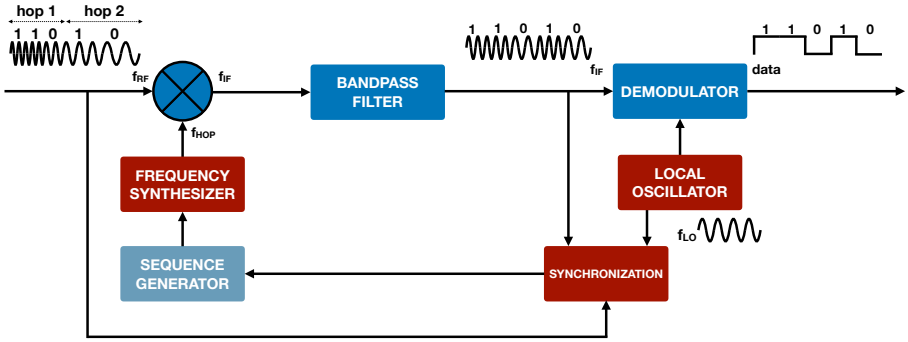


Figure 2.6 – FHSS demodulator mixing – *dehopping* – the modulated data on the analog waveform up to the carrier frequency according to the generated sequence.

generated inside the transceiver, typically with the help of logic functions (XOR, AND) and shift registers based on the input data - stored hashing keys, stored device ID, clock, etc. [21]. This sequence is known only to the devices, which are communicating with each other - in our case the drone and RC controller. There must be some a priori shared-knowledge of this sequence, for example, an initial pairing of the devices[4].

Generally, this sequence can be arbitrarily long, thus impossible to estimate before the communication of the two FHSS devices ends. Thus waiting and trying to estimate the whole pseudo-random sequence is not recommend.

Frequency Syntesizer The purpose of the synthesizer is to generate a waveform, which frequency corresponds to the frequency pseudo-random sequence. The synthesizer is constructed as a voltage controlled Oscillator (VCO), which is driven by the local oscillator together with phase-locked Loop (PLL) to keep the output frequency as accurate as possible. The other option is to deploy direct digital synthesis (DDS), where the FHSS waveform is generated in the digital domain. Important note is that the frequency synthesizer has certain reactional time, which means a limitation for the possible speed of the hopping process.

FHSS Pulse The FHSS pule has several properties worth explaining. First, we declare that we chose pulse at one frequency channel – we are aimed at one carrier frequency from the hopset. Looking at the figure below2.7, we notice that the moment when the channel is fully occupied by the signal is the ”dwell time”. Within this interval, the energy in the channel is maximal and the modulated data are being transmitted. However, thanks to the RF components limitations, the interval of the change in the energy cannot be infinitesimal. The *rise time* and the *fall time* transient states are always present.

Fast or Slow fhss In general, the hoprate could be higher than the symbol rate defined by the data rate of the sent message and chosen modulation. In

[4]Most preferably in the environment, where nobody else can listen to this sequence.

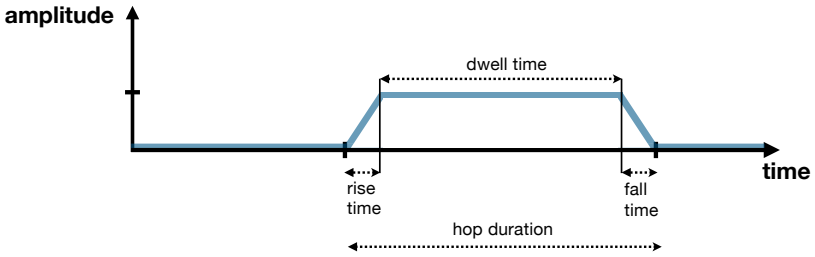


Figure 2.7 – Illustration of the FHSS pulse, which has a certain rise and fall time at the given frequency channel hop [22]. Due to the transition intervals, FHSS systems must consider them and let the transmitter/receiver retune to the new frequency.

such a case, the period time of the hop T_{hop} is shorter than the symbol period, and we call this scenario fast FHSS. The advantage of using fast spreading is that with proper data message encoding, the fast FHSS is more resilient to the fading and noisy channels, possible tone jammers. Also, the detection is more difficult for potential detection systems. Nevertheless, the transmitted waveform of the fast FHSS is not spectrally compact – a large channel bandwidth is occupied, thus, th slow FHSS is usually preferred as stated on page 133 in [19].
With the slow FHSS, several channel symbols are transmitted within one hop, i.e., the hop rate is lower than the symbol rate. In the scenario, where we want to identify such FHSS link, the slow FHSS seems to be easier to detect. The fast FHSS signal will not be investigated.

2.4 Operating Bands

As we were talking about the frequency spectrum as a natural resource, its allocation is conditioned by the international and national regulators. Most of the spectra are allocated to military purposes. Some of the spectra are licensed, e.g. the telecommunication operators must pay for the portion spectra their telecommunication technology – GSM, UMTS or LTE – occupies. The FM radio and DVB-T television broadcast bands are also subjects of a licensing fee.

However, there are still chunks of spectra which are free to use for any application. Those are the Industrial, Scientific and Medical radio bands (ISM). The power, with which your transmitter can radiate at those frequencies and the time duration of the radiation, is regulated by the International Telecommunication Union (ITU). Since those bands are attractive for lots of applications, those regulations are necessary to provide fairness of use.

For the UAV's FHSS radio communication link, not all of those ISM bands are suitable. If we look in the complete ISM band table[5], we would see that some of the bands are too narrow – only several kHz wide. Therefore you can find the selected bands, where the potential UAV communication systems may be operating in the table below 2.1.

[5] taken from [23]

frequency range [MHz]	center frequency [MHz]	bandwidth [MHz]	region
433.05 – 434.79	433.92	1.74	Europa, Africa, former Soviet Union, Middle East west of the Persian Gulf, Iraq
902 – 928	915	26	North and South America, Greenland, eastern Pacific Islands
2400 – 2500	2450	100	worldwide
5725 – 5875	5800	150	worldwide

Table 2.1 – ISM bands.

In the table, you can also see, which different technologies operate within each band, thus potential sources of interference. Especially the 2.45 GHz band is overfull. Another fact is that not all the ISM bands are available over the whole world. For example, the 915 MHz band is reserved as a licensed band for telecommunication (GSM, LTE) in Europe.

2.5 UAV Radio Communication Link Summary

Having read all the previous sections, we now sum up the characterizations we can presume about the unknown UAV's radio link. Those assumptions will be our starting point for developing the detection system. Since we will be aiming at real-time detection as we cannot rely on the simple hopping pattern[6], the most important parameters for us are the hop duration and current hop frequency. Having made the assumptions, we also considered the regulations for the ISM bands. For Europe, those regulations can be found in [24] and [25]. Another useful document we studied in order to learn more about options of FHSS signal deployment in ISM bands can be found in [26].

Hopping Band Since the vendors aim at a low price of their commercial products, using licensed band is not an option. Together with the facts, that the FHSS requires sufficiently wide band from its nature and electromagnetic wave propagation issues, we can expect that the FHSS system will be operating in 434 MHz ISM band, 2.45 GHz ISM band or 5.8 ISM band.

Data Rate The driving commands for the drones are typically very simple and do not require high data rate. That means we will be exclusively dealing with the uplink – data from the RC controller to the drone. However, the data transfer must be highly reliable. Thus we can expect considerable redundancy of data to improve the probability of correct data reception. That is mainly a task for the encoder block.

[6] remember Bluetooth, where the sequence is pretty complex and its period last one day, viz. [21]

Hop Duration Inspired by the Bluetooth, we can expect that the hop duration will not be shorter than $100us$. We rather expect values around $1ms$ of the dwell time.

Modulation Assuming non-coherent detection, thus lower transceiver cost, we can expect that some of the FSK modulations will be deployed. Since operating in the ISM bands, the manufacturer should implement the modulation decently to avoid spectral leakage, hence creating interference within the ISM band is not desirable. That leads us to CPFSK, GFSK or GMSK modulations. Like in Bluetooth, an option of DQPSK modulation formats are also possible

Frequency Channel Bandwidth Despite the drone commands are simple, the coding process increasing the redundancy will increase the data rate. With the FSK modulation, we can expect the channel bandwidth from 0.5 to 2 MHz.

Hop Frequencies The set of the frequencies, which the transmitter can hop, is proportional to the selected hopping band and channel bandwidth. We may expect that around one hundred frequencies for the carrier waveform can be deployed by the UAV communication system.

Radiated Power As we can imagine that the commercial drones can be controlled at a distance of several units of kilometers in an open space area, considering the Bluetooth output power and the regulations, we can expect the output power of UAV's controller transmitter around 20 dBm (100 mW).

Channel Additive white Gaussian noise channel assumed. Only one FHSS signal is present. No fading is considered, later in our project, we will try to encounter ubiquitous WIFI interference.

Chapter 3

Software-defined Radio

In the following chapter, we introduce the concept of Software-Defined Radio (SDR), which is becoming more and more popular these days. We will take all the advantages, which that concept provides, and we will be using the commercially available state-of-the-art SDR device in our project.

Apart from the conventional radio transmitters and receivers, which functionality cannot be changed once they are produced, SDR offers flexibility to set a large variety of parameters and functions of the transceiver, for example, a center frequency and a width of the received band, modulation format, amplification of the signal, etc. This makes the SDR especially attractive in the area of research and development. As the SDR term refers, the reconfiguration is done with the help of software, which any user can write and design according to his needs. Thanks to that, the transmitted and received signals are processed in the digital domain, with the help of the software we will design.

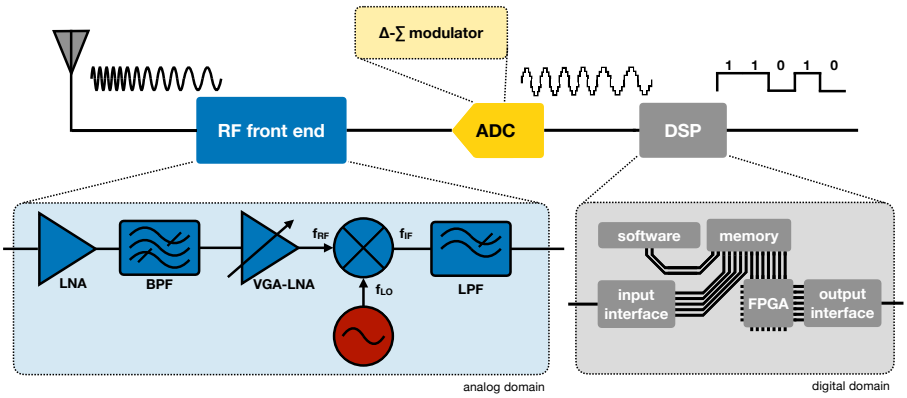


Figure 3.1 – Typical architecture of contemporary software-defined radios.

3.1 RF Front-end Architecture

Recalling the Nyquist theorem, the sampling frequency should be at least twice as high as the maximum frequency included in the bandlimited signal which is being received.

$$f_s = 2 \cdot f_{max} \tag{3.1}$$

where the f_s is the sampling frequency and f_{max} is the maximal frequency contained in the received signal. The sampling frequency directly determines the sample rate of the ADC/DAC. *Remember, the received signal is typically complex. As we will see later, we can split the signal into real and imaginary part and sample it with the* $f_s = f_{max}$.

Thus, with the RF carrier waveform at 2.45 GHz, we would need ADC with the minimum speed of 4.9 Gsps. Such ADC converters already exist, however, they may have a poor resolution and have considerable power consumption. Hence, the concept of mixing down the bandlimited signal down first and then sampling it is still broadly used. Most of the SDR's devices mix the signal down directly to the baseband. Such an RF receiver's architecture is called homodyne.

3.1.1 Homodyne

A homodyne receiver, also known as direct-conversion receiver or zero-IF (intermediate frequency) receiver, employs local oscillator (LO) and mixer to shift the bandlimited signal down to the baseband, where the carrier waveform frequency equals to zero.

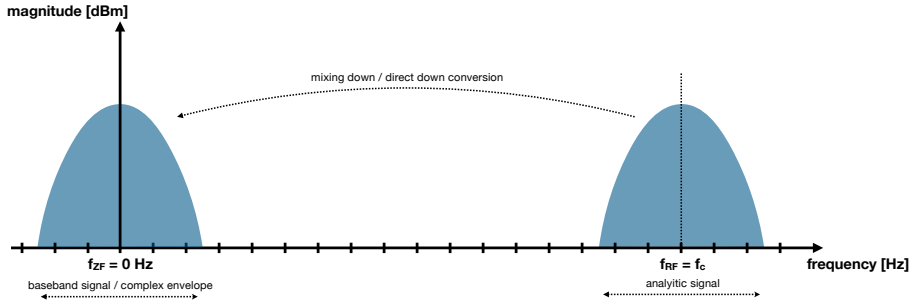


Figure 3.2 – Direct down conversion – mixing down – the signal from the carrier frequency directly to the baseband.

3.1.2 Homodyne Characteristics

The typical RF front-end architecture of the homodye is envisioned in the figure 3.3. The homodyne front-end architecture is attractive for many applications. Due to the significant analog part complexity reduction, the homodyne is less expensive than traditional and until recently very popular superheterodyne. On the other hand, the zero-IF receiver still does not reach the superior qualitative parameters of the superheterodyne transceiver. Technology makes progress

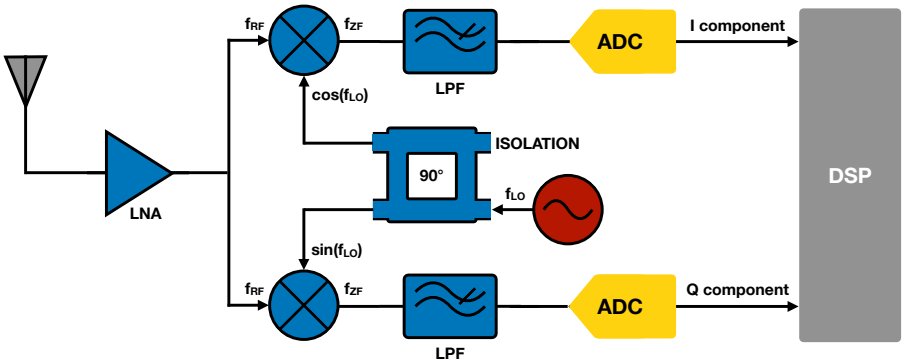


Figure 3.3 – Schematic description of the architecture of the homodyne RF front-end, commonly used in the software-defined radios.

every day, hence the difference is becoming less and less meaningful. Comparing to the superheterodyne, the homodyne is efficiently implementable on a chip, thus the receiver can be much more compact. Therefore, the zero-IF receiver is mostly popular in the conventional applications – WIFI/LTE/Bluetooth modems, DVB-T tuners, so as in the software-defined radio we use in our project.

Furthermore, the homodyne architecture allows higher flexibility of configuration than the superheterodyne one. Since the dedicated analog bandpass filters, required in superhet architecture, are reduced to the minimum or absent, the homodyne can be readily tuned to a specific received signal bandwidth and its carrier frequency.

Homodyne Drawbacks: The homodyne concept also suffers from the following problems we must know before we start using it. Firstly, since the mixer's gates are not perfectly isolated, the local oscillator's signal leaks to the other analog circuits. The leaked signal is then radiated by the receiving antenna. That may cause undesirable interference emitting from the receiver. If the power of the leaked local oscillator's signal is not suppressed efficiently, self-mixing of such a signal occurs. Then, it saturates the amplifiers at the front-end and it adds a DC bias to any received signal. This will result in the significant peak in our spectrum at the 0 Hz frequency. To overcome this problem, the design can include the compensating circuits. The other option is to compensate, or filter the DC component in the digital domain.

The second typical issue is the imbalance of the I and Q components. In the ideal scenario, the LO signal is being shifted exactly by the 90 degrees. Nevertheless, with a realistic phase shifting component[1], the phase shift is not perfectly 90 degrees all the time. In fact, the higher the working frequency range should be, the more difficult is to keep the shift exactly at 90 degrees. This will cause an image signal at the received spectrum. In other words, receiving 100 MHz bandwidth, with the signal at 80 MHz, the mirrored image[2] will appear at 20

[1] for example a branch coupler
[2] sometimes called a ghosting signal in a slang of SDR users

MHz. This drawback can be again partially compensated in the digital domain through a calibration process.

Last, as every passive component is a source of noise according to formula 3.2. Also, every active component – a semiconductor – creates additive shot noise and flicker noise. That degrades the quality of the signal. A curious reader may be interested more in receiver architectures, hence we recommend to go through [27]. Limitations of a zero-IF architecture are discussed in [28], [29] and [30].

3.2 SDR Programming

As we understood, the way how to tell – programme – the hardware how to demodulate the data from the signal, which frequency band we want to tune, etc., is through the software we need to write. The written code is then performed by the processing unit that controls the SDR device and processes the data in it. According to their functions and the ways how the processing units can be programmed, we can classify them into two categories. The first class represents the microprocessors, the second one Field Programmable Gate Arrays (FPGA).

3.2.1 SDR Programming - FPGA

An FPGA consists of configurable logic blocks (CLB), which allows a programmer to define an arbitrary logic function. Those blocks are connected via a programmable interconnection matrix. If a complex logic function is desired, more logic blocks are connected together to perform such a logic function. The logic blocks are then connected to the programmable input/output (I/O) blocks interfacing with the external components like the ADC, or DAC, communication bus controller (PCIe, USB), etc. The configurable logic blocks (CLB) consist of Look-Up-Table (LUT), D-flip-flop circuit and multiplexer. The LUT is a static memory part, where any logic function can be stored. The D-flip-flop provides a possible synchronization of the LUT output to the clock signal, which we will talk about in the next section. The multiplexer then selects if the resulting output should be synchronous with the clock signal.
Since many algorithms coded in the software require complex computations like additions and multiplications, the FPGA architecture is also usually equipped with specialized circuits for efficient adding and multiplying operations – DSP blocks. For efficient data storing, block RAM circuits are implemented.

Clock Signal The clock signal determines the operating frequency of each logic block and memory elements; the whole FPGA chip. The clock is distributed

$$P_n = kTB, \tag{3.2}$$

where k is a Boltzman's constant, T is environment temperature in Kelvins and B is the bandwidth of the received signal.

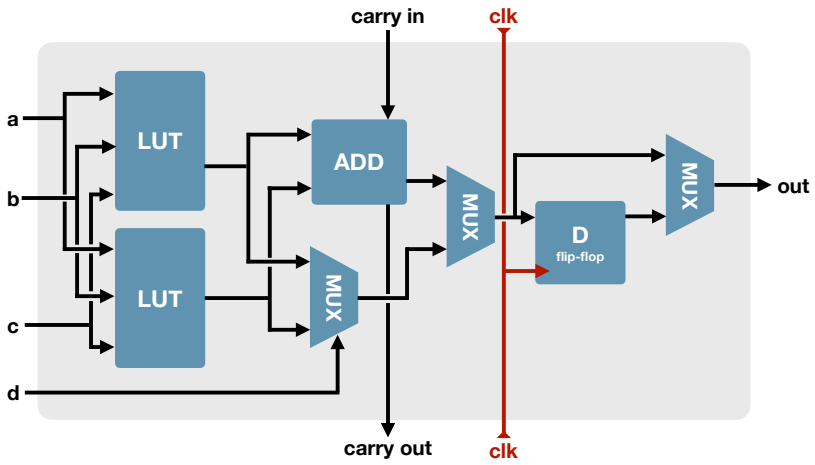


Figure 3.4 – Simplified scheme of the configurable logic block contained inside the FPGA architecture.

to the single components via the interconnection matrix. From this, we can learn that different part of the FPGA can be operating at different frequencies. The purpose of the clock signal is to provide synchronization, to keep the logic operations running synchronously in parallel.

FPGA Programming Procedure To implement the algorithm inside the FPGA in our SDR, we can choose from plenty of options. First, the algorithm functionality must be described in the corresponding language. Common languages are VHDL, Verilog possibly its enhanced version SystemVerilog. These languages belong to the group of Hardware Description Languages (HDL). Furthermore, a translation from familiar higher-level languages like C, MATLAB or LABVIEW is also possible. Technically, we will be using such a translator as we will design the programs for FPGA in the LabVIEW graphical environment. Nevertheless, the output of any FPGA programming environment is always a bitfile, which is loaded into the FPGA and which configures the blocks inside FPGA by interconnecting them as designed.

Host SDR Programming With today's fast interfaces like PCIe or USB 3.0, we can control the SDR from and forward the sampled data to the desktop host PC. Nevertheless, there are still limitations. The main problem is the amount of delivered data, which is related to the chosen bandwidth of the signal received by the SDR RF front-end and th resolution of the ADC/DAC. Even with the high-speed interfaces, forwarding the data is difficult.

VHDL = Very High-Speed Integrated Circuit Hardware Description Langugae

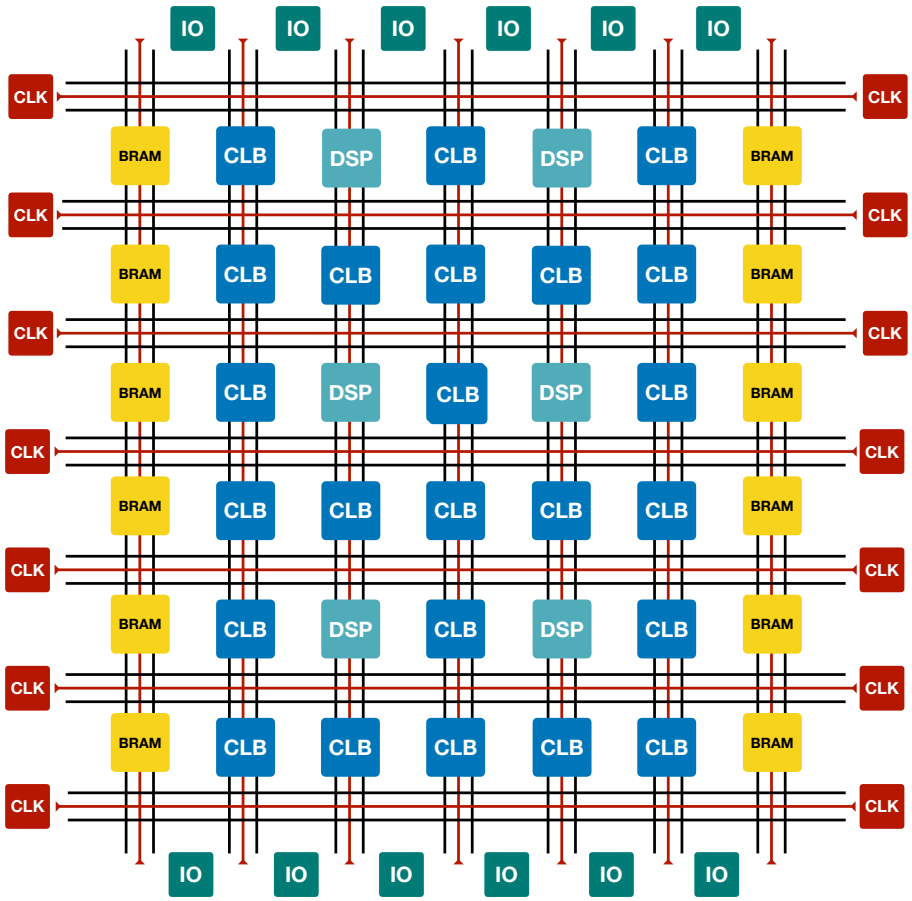


Figure 3.5 – Architecture of the FPGA including logic blocks, RAM blocks, dedicated multipliers circuits and DSP blocks. All the inside blocks are interconnected via the configurable matrix. The interconnection matrix is interfaced with input/output blocks. To each block, the clock signal coming from frequency synthesizer is distributed.

3.2.2 LabVIEW Communication Design Suite

In our work, we develop the design in the LabVIEW Communications System Design Suite 2.0 from National Instruments. The algorithms are described at a graphical level. In this programming environment, we can devlop our own algorihtms running on host PC, or programme an FPGA applications. The FPGA programming is limited since we cannot control all the traditional steps of FPGA programming as depicted in 3.6.

The environment allows us performing the behavioral simulation of our FPGA program. However, the rest of the FPGA development process is done automatically by the Design Suite. As we will see later, the National Instrument's software-radios have Xilinx's FPGA integrated. Therefore Xilinx's Vivado Design Suite is embedded inside the NI's LabVIEW Communication environment. We can conclude that with the LabVIEW graphical description of the algorithm, the

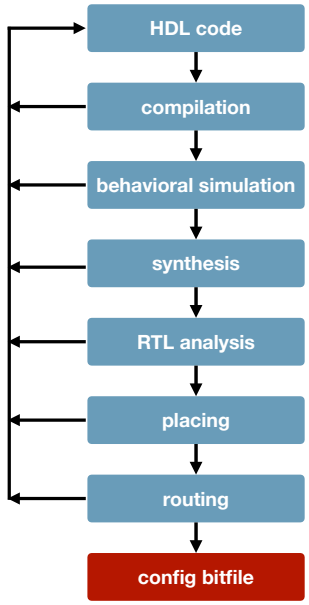


Figure 3.6 – Process of developing FPGA design. *As you can see, the process of* FPGA *programming includes many steps. It may happen easily that the last step doesn't fulfill requirements, then we may need to start from the scratch. Thus, the* FPGA *programming is sort of burdensome.*

LabVIEW interpretation is translated to the HDL code. That HDL code is forwarded to the Xilinx Vivado. Moreover, we can exploit the IP blocks provided by the Xilinx in the LabVIEW Communications Suite, which is a feature we will be using at the end of our project.

3.3 State-of-The-Art Software-defined Radio

In our project, we use a state-of-the-art software radio manufactured by National Instruments, type USRP[3] *2953r* with the CBX *1200-6000 MHz* RX/TX *120 MHz* as installed RF front-end board. This gives us a transceiver with two RX and two TX channels. The transceiver is built as a homodyne, which working principle, advantages and drawbacks were described in the section above. As a digital data processing unit, our SDR is equipped with an FPGA *Kintex XC7K410T* chip manufactured by Xilinx.

Our SDR is then connected to the switch via the PCIe x4 wired interface. Practically, the switch would not be necessary, nevertheless, the USRP we are using

[3]Universal Software Radio Peripheral

The original developer of the USRP devices, Ettus Research, was acquired by National Instruments NI in 2010 [31]. Hence, in most of the cases, certain series of transceivers sold by Ettus and NI companies are nearly identical. E.g., model X310 sold by Ettus corresponds to our USRP 2953r.

Figure 3.7 – USRP 2953r 2x2 SDR transciever

is part of a system with many deployed USRPs. In our project we work only with one USRP device though. From the interface switch, the data are forwarded to the bus connected to the host PC. This last connection is realized via the PCIe x8 through an optical fibre. The host computer is equipped with the Intel Xeon CPU with eight cores, assisted by 24 GB of RAM. The operating system of the host PC is Windows 7 Enterprise. The whole setup is depicted in the figure 3.8.

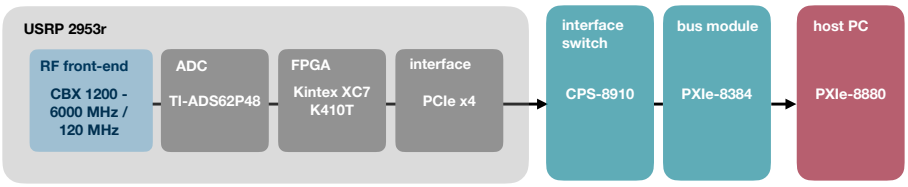


Figure 3.8 – Deployed SDR setup in the project.

We use the host PC to control and set the USRP, which we use only as a receiver in our project. The USRP has to be properly set, i.e., the received band and its width. This is accomplished in the *LabVIEW Communications System Design Suite 2.0* program we mentioned in the previous text. As it was stated, this programming environment allows us to develop signal processing algorithms. In general, those algorithms can be run either on the host PC or on the FPGA inside the USRP device. Nevertheless, this environment always plays an irreplaceable role during the initial setting of the our universal radio receiver, i.e., tuning the frequency band and its width, selecting the receiver's channel.

To learn more about all the equipment we are using in our project, we suggest to go through following documents: RF-front end in details including used components [32] and its schematic [35]; specification of the USRP 2953r [33] and [34]; more about dedicated FPGA [36] and [37]; more about the CPS-8910 switch [38], specifications of the PXIe bus module in [39]; and finally about the host PC in [40].

CPU = central processing unit
RAM = random-access memory

parameter	value	reference
omni-directional antenna	3dBi at 2.45 GHz	Vert2450
frequency range	1.2 – 6.0 GHz	[32], [33]
max. bandwidth	120 MHz s	[32], [33]
ADC resolution	14 bit	[33]
dynamic range	88 dB	[33], [34]
max. FPGA – PC throughput	800 MB/s	[34], [33]

Table 3.1 – Selected parameters, which need to be considered as a hardware limitations of our project.

3.4 Software-defined Radio Summary

Having read chatper 3, we understood the benefits and limitations of the SDR we use in our project. With the USRP 2953r, we will be able to receive a signal with the bandwidth up to 120 MHz in the frequency range from 1.2 to 6 GHz. Following the information and assumptions made in chapter 2, we conclude that sensing a drone remote controllers' signal should be viable with our device. Nevertheless, we anticipate troubles with DC component caused by the local oscillator leakage. Also, mirrored images of the signal in the spectra are expected.

Chapter 4

Time-frequency Analysis

In chapter 2, we were talking about Frequency-Hopping Spread Spectrum signals which characterize our UAV radio communication link to be detected. We understand that those signals change their frequency during the time, thus, we defined a time interval – the hop duration – during the carrier frequency of the transmitted signal remains constant. Therefore, apart from other communication signals used in WIFI, LTE, etc., FHSS signals are considered as non-stationary ones – their frequency is not stationary within the time. Since we are aiming at a parameter estimation of unknown FHSS signals, mainly the hop duration and the instantaneous frequency of the carrier, we need to analyze them in both domains – in the time domain and also in the frequency domain.

Traditional time or frequency analysis techniques are not suitable for our FHSS signal. Fortunately for us, many astrophysicists, mathematicians, geophysicists, signal processing engineers, or theoretical physicists introduced time-frequency signal analysis techniques helping us to overcome the limitations of classical methods and analyze the signal simultaneously in the time and the frequency domain [41], [42], [43]. Nowadays, as the amount of data, which needs to be processed is rapidly increasing, compressive analysis techniques are becoming tempting [44], [45], [46], [47], [48], [49], [50], [51], however, we will not deal with them within this project.

On the following lines, we provide a comparison of selected time-frequency analysis (TFA) techniques used on a simulated signal in the MATLAB environment. Also, we briefly introduce an intuition how the techniques work and what are their advantages and drawbacks.

4.1 Simulation Overview

To introduce the TFA techniques, we prepared a small simulation of an FHSS signal in MATLAB. Since we do not have any measurement record of such signal at this moment, we found an inspiration in a Bluetooth technology [21] and used its common parameters to generate the FHSS signal. Afterwards, Short-time Fourier Transform, Wavelet transform and Wigner-Ville distributions were

used for analysis. The code for the simulations script can be found in appendix A.1.

Simulated radio link characteristics	
band	ISM: 2400 - 2500 MHz
frequency channel BW	1 MHz
number of freq. channels	100
hopping rate	1600 hops per second
hop duration	625 us
modulation formats	pure FSK
bit rate	1 Mbps

Table 4.1 – Parameters of the simulated signal.

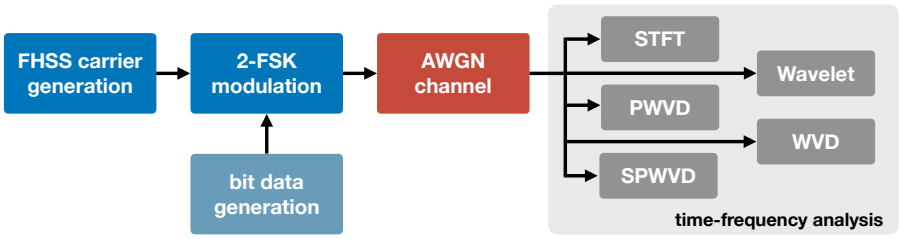


Figure 4.1 – Block diagram of the simulation.

4.1.1 Generated Signal

First, a hopped carrier is generated according to the parameters in the table 4.1. To provide a meaningful comparison, see the hopping pattern in the figure 4.2 below. Note that the hopped carrier is illustrated by a rectangle boxes for better readability. The carrier is, of course, only single frequency, which would correspond to lines. The channel bandwidth of each hop is determined in the next step by the used modulation and data rate.

Next, the carrier is modulated by the data signal. As a modulation format, simple FSK[1] is used. The modulation index κ is equal to $\frac{1}{2}$ and we do not consider a continuous phase, therefore abrupt change of phase of the modulated signal are expected[2]. After modulation, the signal passes a channel where only additive white Gaussian noise (AWGN) is present. For the simulation, no interference is assumed.

[1] frequency shift keying

[2] Later, we will learn that those abrupt phase changes are one of the causes of the spectral leakage when the Fourier analysis is applied.

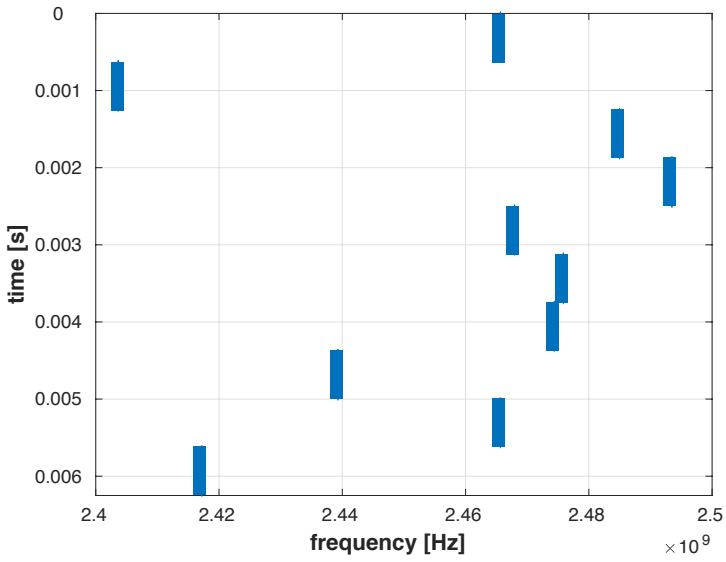
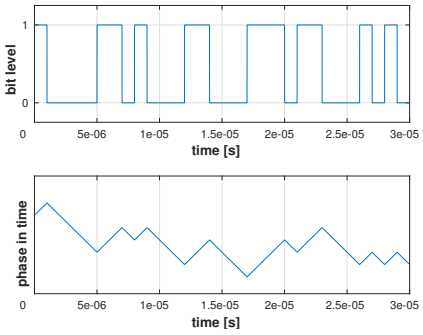
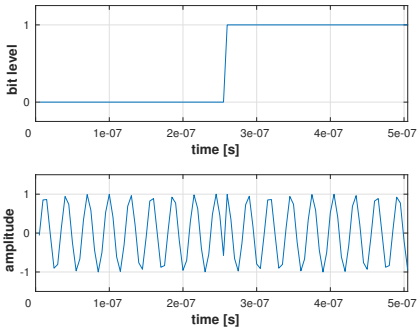


Figure 4.2 – Pseudo-randomly generated frequency hopping pattern used within this simulation for comparison of several TFA techniques within this chapter. We see ten hops over 100 MHz band. Total time of the simulated signal is then 6.25 ms.



(a) Data signal represented by randomly generated bit sequence. No communication data structure (header, payload, coding, etc.) was implemented. Also, the square data waveform corresponds to the phase change.



(b) Detail of the modulated signal. Notice the abrupt change of the phase of the signal, when the bit is being changed from 0 value to value 1.

Figure 4.3 – Detail of the generated FSK modulated signal.

4.2 Short-time Fourier Transform

The first technique we present is the simplest and fastest regarding computational operations. We spill the beans by declaring that this technique will be used later in our project for FHSS radio link detection. The short-time Fourier transform roots (STFT) to the Fourier transform for signal transfer between the time and frequency domain – the Fourier transform (FT). We devote the next chapter to describe the properties and characteristics of the (discrete) Fourier transform.

The mathematical formula describing the STFT spectrum computation of the continuous signal fraction is:

$$S(\mu,\omega) = \int_{-\infty}^{\infty} s(t)w(t-\mu)\mathrm{e}^{-\mathrm{j}\omega t}\,\mathrm{d}t \tag{4.1}$$

where $s(t)$ is the analyzed signal, $S(\mu,\omega)$ is the analyzed time-frequency result, w is the short-time window, e is the Euler's number, j is the imaginary number, ω is the angular frequency, t is the time axis, and μ is the time shift in the direction of time axis

The main disadvantage of the STFT is the necessary trade-off between the time and frequency resolution implied by the equation 4.2.

$$\Delta f = \frac{f_s}{2N} \tag{4.2}$$

The f_s stands for Nyquist sampling frequency, and the size of the short-time interval in time-samples is denoted by N. We distinctly see what happens if we let the signal bandwidth constant and play with the number of samples N. The frequency resolution of the computed spectra is increasing with the higher number of N. Nevertheless, we must realize, that those samples must be received first before the spectrum computation can start. In other words, we must wait till all the N samples are collected. As the ADC speed is fixed, it introduces a time delay leading to decrease of the time domain resolution of the resulting spectrogram. In general, this trade-off leads to the uncertainty relationship principle.

4.2.1 The Uncertainty Principle

The uncertainty relationship principle roots to the quantum mechanics where it was formulated by theoretical physicist Werner Heisenberg. To simplify the idea behind that Heisenberg's uncertainty principle states that if you precisely know the momentum of a particle, you cannot simultaneously know its position and vice versa.

Presenting the uncertainty principle within a signal time-frequency analysis, it can be expressed as the inequality:

$$\Delta f \cdot \Delta t \leq 1 \tag{4.3}$$

where the Δf is the frequency resolution, and t is the time resolution.

4.2.2 Spectrogram

As the signal waveform is continuing in time[3] – being constantly received, the STFT technique gives us consecutive spectrum snapshots in time; that means time-frequency analysis.

Favorite interpretation of the power densities belonging to each frequency component of the individual signal fractions is a spectrogram. The spectrogram is a graphical representation of the power strength of the signal at a certain frequency at a certain time moment. It is a two-dimensional graph, where one dimension represents the time axis and the second one the frequency axis. The information about the signal power - the magnitude - at each time-frequency position is scaled into colors.

The Fourier transform used in STFT time-frequency analysis preserves the complex-valued character. In our work, we are mainly interested in the magnitude[4] of the received signal.

Despite its simplicity, the STFT is powerful time-frequency analysis technique. If the short-time interval – the length of the fraction taken from the signal is defined properly, which is related to the fact, how rapid the signal is in terms of non-stationarity, the STFT can provide decent results. Since the STFT applies the Fourier transform, respectivelly discrete Fourier transform, to compute the spectrum components, the STFT's appreciatable advantage is that it can be computed swiftly since it can exploit the Fast Fourier Transform (FFT) algorithm, which we will talk about in the next chapter. This makes STFT an ideal candidate for real-time applications. It is one of the main reason, why we decided to use the STFT technique in our project. On the other hand, the fundamental limitation of the STFT time-frequency analysis tool is that we always need to trade between the frequency resolution and the time resolution of the spectrogram.

At this point, we go back to the simulated signal 4.1, which passed the AWGN channel. We now analyze the signal with the STFT technique with several different settings. For all TFA 2-D graphs, the y-axis always represents the time in seconds and the x-axis stands for frequency in Hertz. In other words, the STFT spectrogram is built from rows, where each single row represents a DFT spectrum, computed from a finite number of time samples coming continuously from ADC.

When we look at the figure 4.4, we see the STFT resulting spectrogram, where each horizontal row has the length of 8192 taken points. That gives us fine frequency resolution. On the other hand, the time resolution is quite poor. We may agree that for the estimation of the hop duration, such time resolution is enough. However, for sure, we will not be able to distinguish between the bits represented by two frequencies of the FSK modulation[5]. The other drawback of

[3] do not mistake the continuing with continuous. The discrete-time signal sample can also continue in time domain

[4] absolute value of the complex-valued frequency bin

[5] which will not be crucial for our pioneering FHSS signal parameter estimation

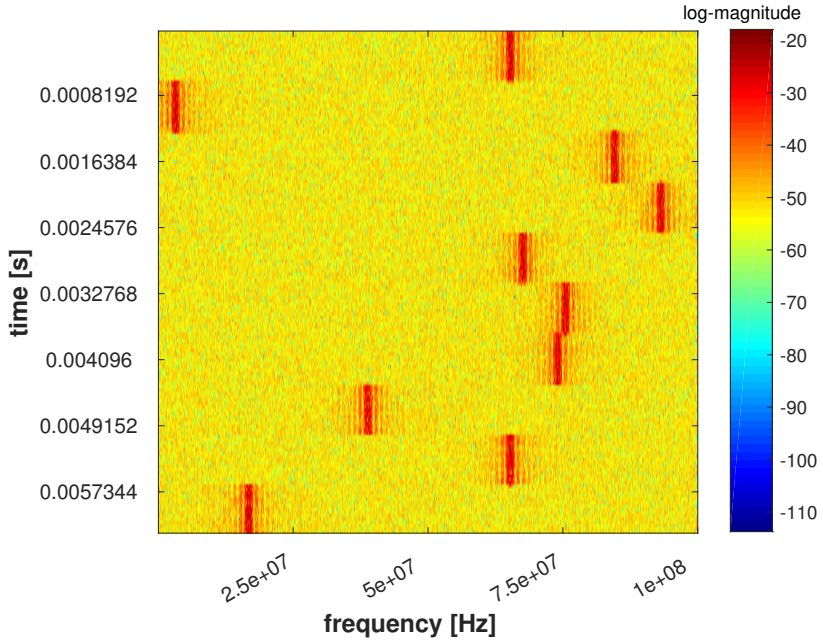


Figure 4.4 – STFT spectrogram where each row corresponds to the 8192 samples in frequency. With the 100 MHz sampling rate we are using, the frequency resolution of that row is then 12 kHZ and its time domain resolution is equal to 81.92 us.

this fine frequency resolution is that the larger the DFT spectrum frame shall be, the longer time its computation needs.

Another situation is envisioned below in the figure 4.5. At this point, we were impatient and took only a few samples (comparing to the previous spectrogram) before we started computing the spectrum – we took only 128 time samples to compute 128-point DFT frame. That results in the fine time resolution and poor frequency resolution, which may not be acceptable.

With an adequate length of the STFT's window, we can get a spectrogram with a sufficient time and also frequency resolution so that the tones are clearly recognizable and well-localized in time. The figure 4.6 below is an example where we are close to accomplish such a task. Assuming 100 MHz sampling rate, this STFT spectrogram provides a frequency resolution of 200 kHz and 5.12 us resolution in the time domain.

Except for the time-frequency resolution trade-off, the use of STFT brings other problems, which are mainly connected to the discrete Fourier transform properties. We will explain them in the next chapter. For example, a bright reader may have noticed the level of the noise background in the figures 4.4, 4.5 and 4.6. Looking at the pictures again, we clearly see that noise background lowest when we are taking the largest point DFT. This phenomenon is called processing gain and we will be talking about it in the next chapter.

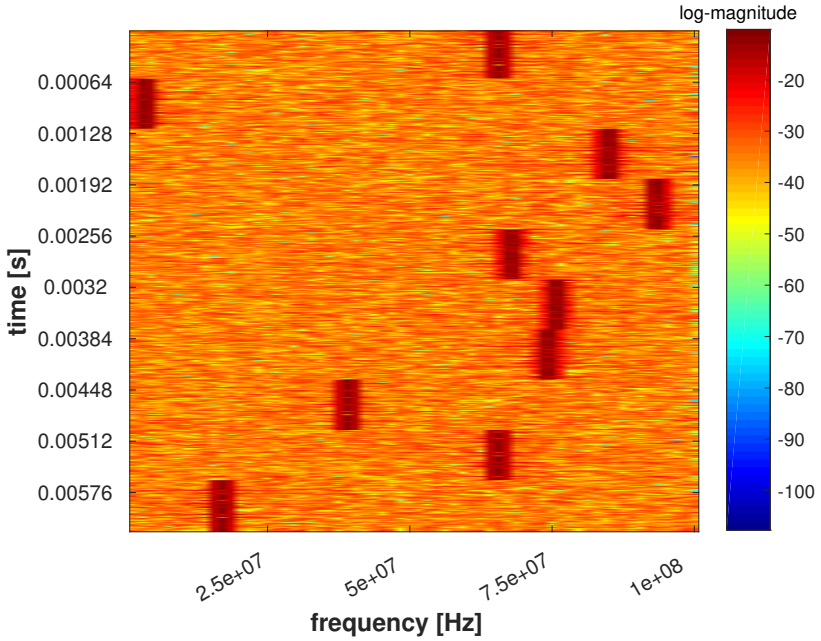


Figure 4.5 – STFT spectrogram where each row corresponds to the 128 samples in frequency. With the 100 MHz sampling rate we are using, the frequency resolution of that row is then 781 kHZ and its time domain resolution is equal to 1.28 us.

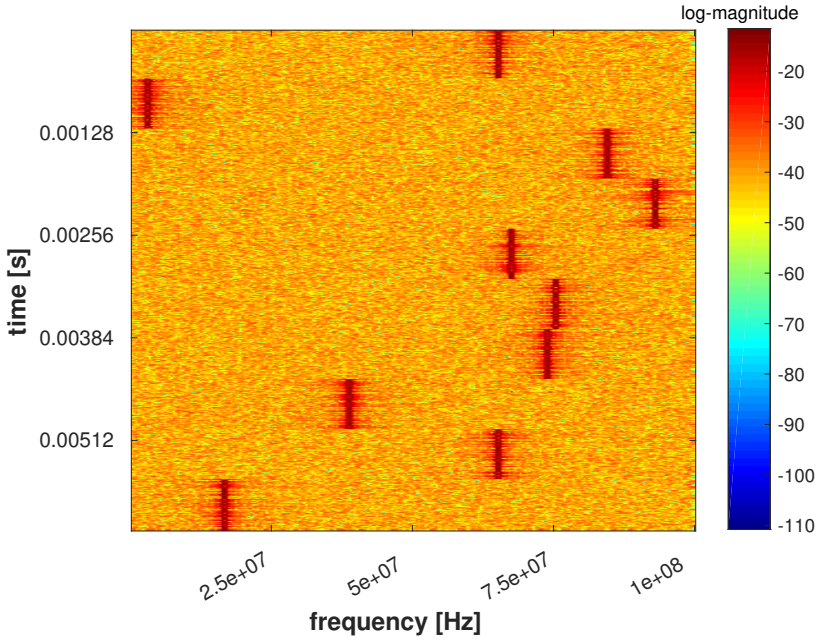


Figure 4.6 – STFT spectrogram where each row corresponds to the 512 samples in frequency. With the 100 MHz sampling rate we are using, the frequency resolution of that row is then 200 kHZ and its time domain resolution is equal to 5.12 us.

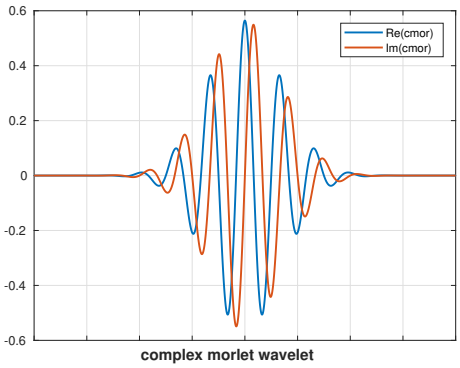


Figure 4.7 – Complex morlet wavelet.

4.3 Wavelet transform

The wavelet transform allows us to extract information about the presence of a frequency component located at a certain time moment. It may be seen that such property must violate the Heisenberg's uncertainty principle. We continue explaining the transform to calm the reader that such behavior remains still impossible.

Before we state the mathematical formula for the Wavelet transform, we recommend a reader to take a look at the STFT equation once again 4.1

$$W_\Psi(\sigma,\tau) = \frac{1}{\sqrt{\sigma}} \int_{-\infty}^{\infty} s(t)\Psi\left(\frac{t-\tau}{\sigma}\right) \mathrm{d}t \tag{4.4}$$

where σ is the wavelet scale, $s(t)$ is the analyzed signal, t is the time axis, Ψ is the used wavelet, and τ is the time shift

Looking at the equation above, we see that comparing to the STFT expression 4.1, the WT is not very different. The main difference is the transform function. While in the STFT, the signal is decomposed to the harmonic functions represented by the $e^{-j2\pi f}$, the WT is decomposed to the scaled function called wavelet. This word comes from French meaning short-wave. There is a whole family of wavelet functions suiting a different kind of time-domain signal.

The wavelet transform computation performs time-scaling of the wavelet function – the wavelet – which is envisioned in the figure 4.7, is being stretched or compressed during the calculation of the WT output. As a consequence, the WT is sometimes called time-scale analysis. By compressing the wavelet, the higher frequency components are extracted from the signal; by stretching the wavelet the lower frequencies are obtained. That will point us back to the trading the time-frequency domain.

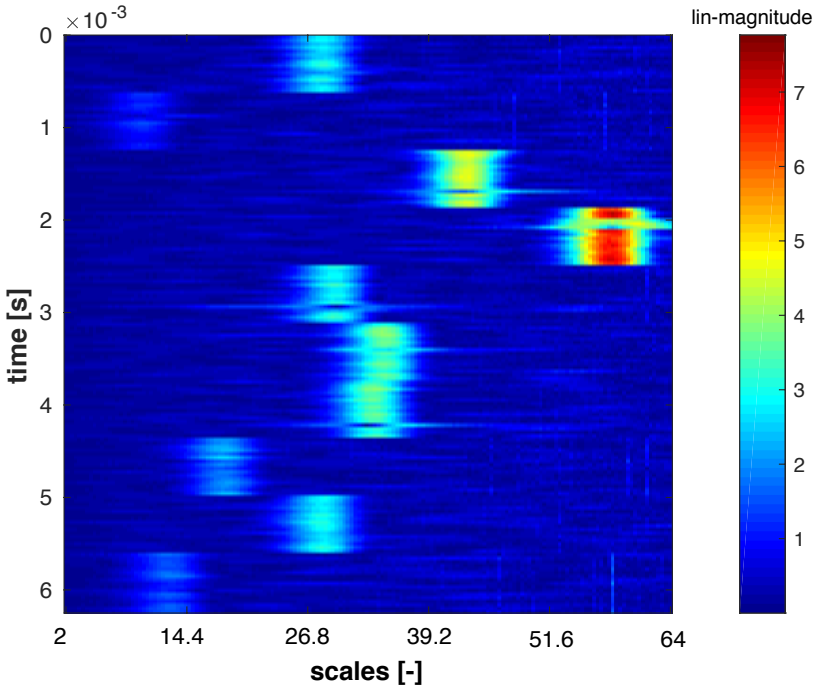


Figure 4.8 – Wavelet analysis of the simulated signal – frequently called scalogram.

As a result of the scaling process, the bandwidth of the signal we are trying to analyze is scaled – the frequency components are not uniformly distributed along the frequency axis as with STFT. That is visible in the figure above 4.8. As a conclusion, we do not believe that a wavelet transform is an practical tool to anaylze our FHSS UAV signal, despite the fact that some work has been done on this topic [52], [53].

4.4 Wigner-Ville Distribution

Many of the publications [54], [55], [56], [57], [58], [59], [60], [61] try to deploy the energy distributions from the Cohen's class [42] in practice to analyze the signal in the time-frequency domain. The often-cited representative is the Wigner-Ville distribution (WVD) and its derivatives. Originally proposed by the Eugene Paul Wigner as a tool for quantum mechanics data processing, the WVD can outperform the STFT and WT techniques in some signal processing cases as Jean Ville discovered.

The main advantage is that WVD can provide a slightly different view on the time-freququency signal representation and bypass the limitation given by the uncertainty principle. The pure WVD is defined as:

$$WVD(t,f) = \int_{-\infty}^{\infty} s\left(t+\frac{\tau}{2}\right) s^*\left(t-\frac{\tau}{2}\right) \mathrm{e}^{-\mathrm{j}2\pi f\tau}\,\mathrm{d}\tau \tag{4.5}$$

where t is the time, f is the frequency, $s(t)$ is the analyzed signal, τ is the value of the shift in the time domain, and * is denotes the complex conjugate

The expression above can also understand as the Fourier transform of an instantaneous autocorrelation function of the analyzed signal $s(t)$. The WVD satisfies many interesting mathematical properties. For curious readers, we recommend going through literature [62], [63], [43].

Another problem which arises with the use of energy distributions like Wigner-Ville distributions is that WVD and later introduced PWVD create interference artifacts called cross-terms[6] The illustration of the cross-terms can be seen in the figure 4.9. The interfering cross-terms artifacts have devastating effect on the result of time-frequency analysis, thus, the pure WVD is not suitable for the FHSS project.

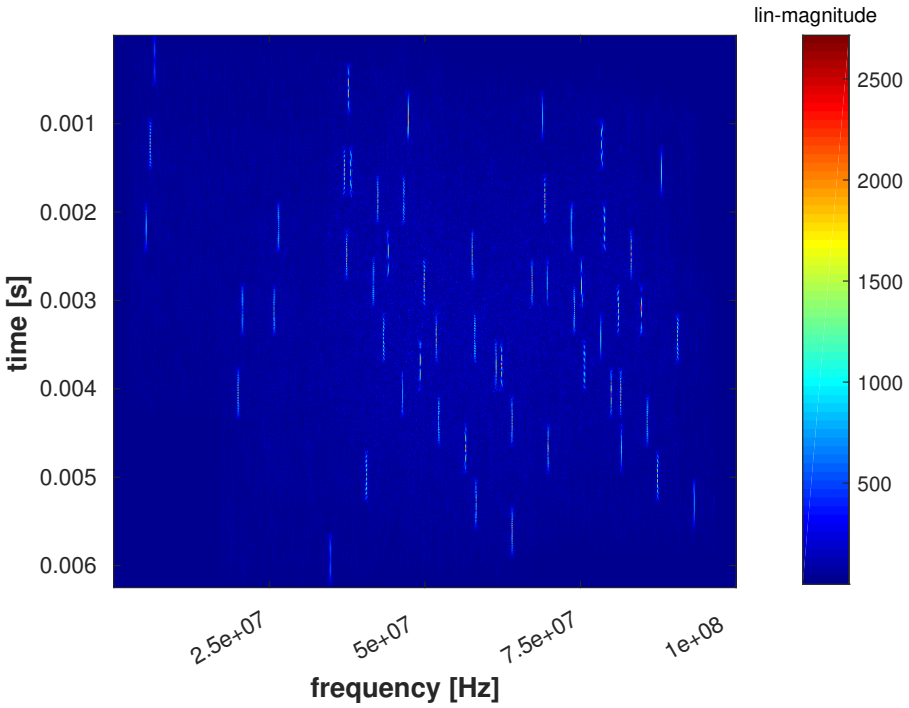


Figure 4.9 – WVD analysis.

[6]Within the scope of this thesis, we will not thoroughly describe the mechanism of cross-terms, however, you can study their effects thoroughly in [63], [43], [42], [62].

4.4.1 Pseudo Wigner-Ville Distribution

Looking at the formulas above, we conclude that they assume an infinite signal interval. In practical applications, that is certainly a problem. Therefore, the idea with the short-time interval from the STFT is used. Only a fraction – a window – from the time-domain signal is taken. Such distribution is then called Pseudo Wigner-Ville distirbution (PWVD):

$$PWVD(t,f) = \int_{-\infty}^{\infty} w(\tau) s\left(t+\frac{\tau}{2}\right) s^*\left(t-\frac{\tau}{2}\right) \mathrm{e}^{-\mathrm{j}2\pi f\tau}\,\mathrm{d}\tau \tag{4.6}$$

where t is the time, f is the frequency, $s(t)$ is the analyzed signal, τ is the value of the shift in the time domain, * denotes the complex conjugate, and w is the windowing function in the time domain

The introduced window points us back to the Heisenberg's uncertainty principle as the time-domain window is equivalent to a frequency smoothing of the original WVD. That means we will again need to think about the time-frequency resolution we want to achieve as we had to do with the STFT analysis. The explanation can be found on page 63 in [63].

We applied a PWVD analysis technique with the 512 samples long rectangular-shaped window on our simulated signal and we obtained results depicted in the figure 4.10

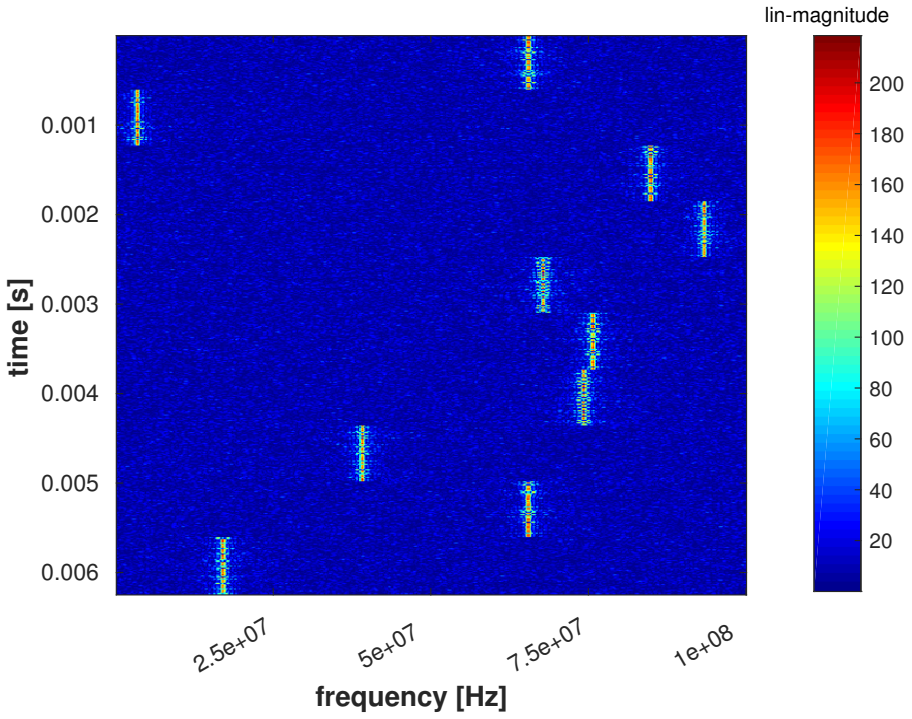


Figure 4.10 – Pseudo Wigner-Ville analysis of the simulated signal. Smoothing window in time domain is set to 512 samples. The shape of the window is rectangular.

4.4.2 Smoothed Pseudo Wigner-Ville Distribution

To diminish more the effect of cross-terms, the PWVD can be smoothed once again. Another smoothing function allows us to control the smoothing process in the frequency as well as in the time domain, but independently to each of those domains. Different shape of the window can be chosen for each domain.

$$SPWVD(t,f) = \int_{-\infty}^{\infty} w(\tau) \int_{-\infty}^{\infty} g(x-t) s\left(x+\frac{\tau}{2}\right) s^*\left(x-\frac{\tau}{2}\right) \mathrm{d}x\, \mathrm{e}^{-\mathrm{j}2\pi f\tau}\, \mathrm{d}\tau \tag{4.7}$$

where t is the time, f is the frequency, $s(t)$ is the analyzed signal, τ is the value of the shift in time domain, * is denotes the complex conjugate, w is the windowing function in time domain, and w is the windowing function in time domain, and the g is the window smoothing in frequency

We need to realize that this additional smoothing process again decreases the final time-frequency resolution, which is noticeable if we compare the figures 4.9, 4.10 and 4.11. Another effect of the smoothing in frequency domain flattens the noise background.

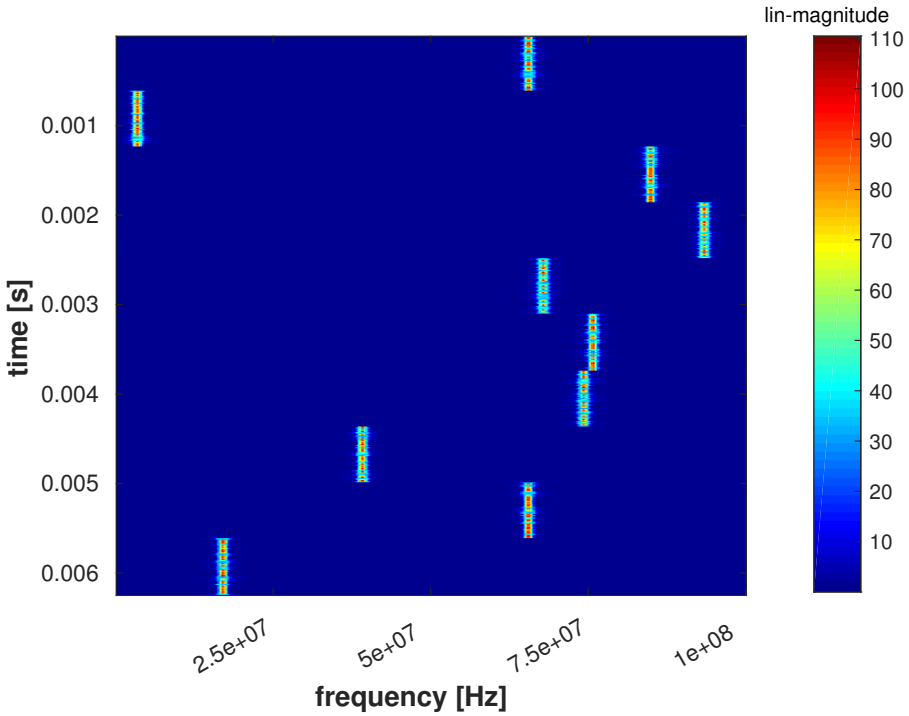


Figure 4.11 – Smoothed Pseudo Wigner-Ville analysis of the simulated signal. Smoothing window in frequency is set to 512 samples and its shape correspond to the hamming window. The time domain window is 512 samples long with rectangular shape.

4.5 Time-Frequency Analysis Summary

Before we make the conclusion, which technique will be used, take a look at the figures 4.12 and 4.13. With the STFT analysis, the estimation of the bit sequence is impossible due to the uncertainty trade-off. With the PWVD and SPWVD, the transients between the bits are visible. This may be caused by the nature of the FSK modulation with abrupt phase changes.

Nevertheless, the main drawback associated with the Wigner-Ville distributions is its computational complexity. In the section discussing the Wigner-Ville distributions, the instantaneous/local autocorrelation is being calculated during the WVD computations. The complexity of calculating an autocorrelation function in the time domain is proportional to *O(N2)*. As we already know, Fourier transform is then applied on the result of the correlation function. Later in the text, we will learn that DFT can be performed by the computationally efficient $Nlog(N)$ algorithm called FFT, which is used in STFT. That complexity plays a main role in the real-time implementation scenario.

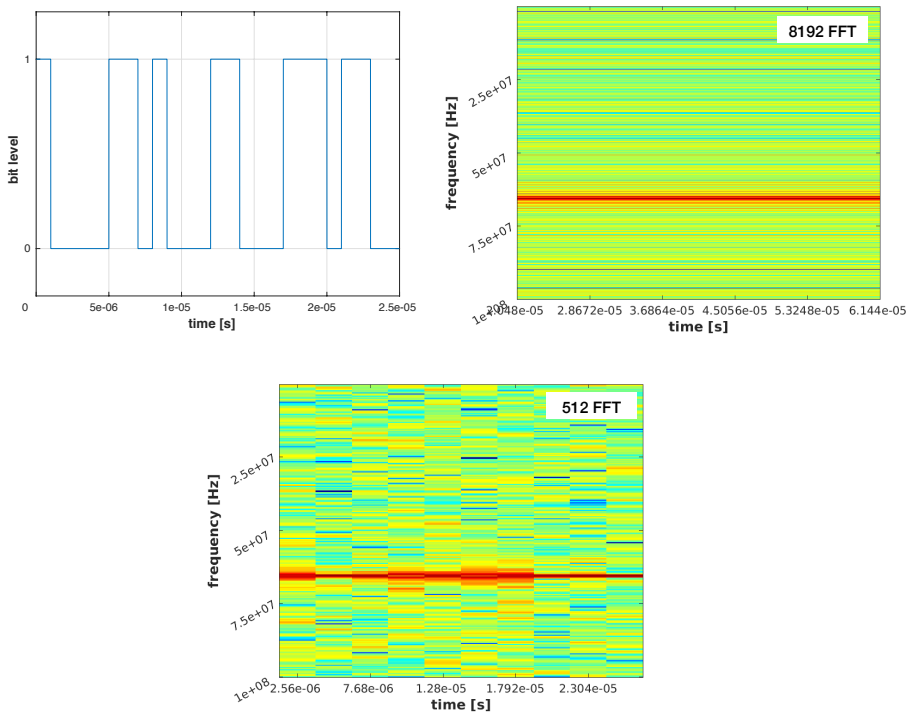


Figure 4.12 – From the left top: bit sequence; smallcapsstft with 8192 points spectrum size, detail. No changes of bits are visible due to poor time resolution; STFT with 512 points spectrum size, detail. Time resolution is still not sufficient to distinct the bits in the spectrum.

Supposing we are going to implement the final algorithms into an FPGA, the computational requirements of each time-frequency technique need to be taken

into an account as FPGA provide only a finite amount of computational resources. It is worth considering that in this context, the time of the computation represents a resource also. Hence, the complex algorithm may not be suitable for real-time applications like we desire.

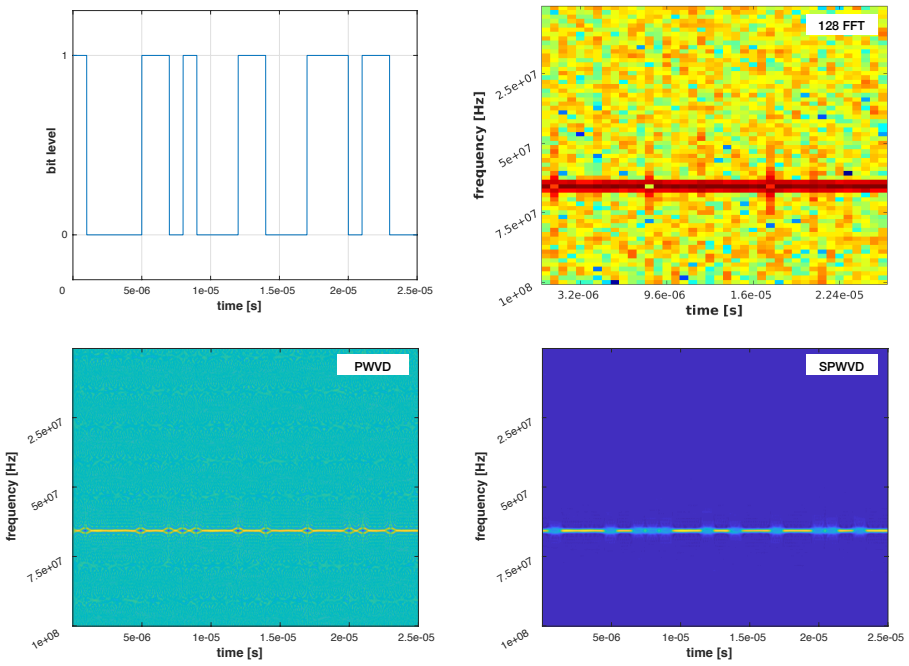


Figure 4.13 – From the left top: bit sequence; STFT with 128 points, detail. No changes of bits are visible due to poor frequency resolution; with an PWVD analysis of an FSK signal, bits changes are distinguishable; with an SPWVD analysis of an FSK signal, bits changes are also distinguishable.

To provide at least a rough comparison of the computation time, we measured the running time of the sections of our script A.1 belonging to each TFA technique. The STFT computational time did not cross a value of 1 second. The PWVD computations took about 40 seconds, and the SPWVD lasted more than 3000 seconds. As we are not focusing on development of efficient algorithms to compute the Wigner-Ville distributions, we conclude that those techniques are not suitable for this project.

Chapter 5

Short-Time Fourier Transform Properties

At the end of the previous chapter, we decided to use the Short-Time Fourier transform as a tool for analyzing the FHSS signal. Since the STFT benefits from all the advantages of Fourier transform, meanwhile it suffers from all the drawbacks, we consider as necessary to characterize the Fourier transform main properties.

The Fourier transform decomposes the signal into periodic harmonic functions – sines and cosines. The profit of the decomposition brings us to the domain, where some of the signal characteristics are easily recognizable – a strength of the frequency component, or its phase.

$$S(\omega) = \int_{-\infty}^{+\infty} s(t)\mathrm{e}^{-\mathrm{j}\omega}\,\mathrm{d}t \tag{5.1}$$

where $s(t)$ is the continuous radio signal, e is Eueler's number, j is imaginary number, $\omega = 2\pi f$ is angular frequency and t is time.

The problem with the Fourier transform is that it assumes continuous signal from $-\infty$ to $+\infty$. Since our signal is sampled, the discrete-time Fourier transform DTFT must be used:

$$S[\omega] = \sum_{n=-\infty}^{+\infty} s[n]\mathrm{e}^{-\mathrm{j}\omega n} \tag{5.2}$$

where $s[n]$ is the sampled radio signal, n is the discrete time axis

Then, selecting only a finite set of samples from continuous signal by windowing the continuous signal is stated as the STFT:

$$S[m,\omega] = \sum_{n=-\infty}^{\infty} s[n]w[n-m]\mathrm{e}^{-\mathrm{j}\omega n} \tag{5.3}$$

where w is the short-time window, and m is the shift in samples defining the size of the window

We understand that the computing the Fourier transform from the finite set of windowed samples is equivalent to the taking DFT of those samples. We recall, that the DFT is only an estimation of the true spectra bringing several limitations, which we now introduce.

5.1 Effect of Windowing

The main limitation of the DFT is the DFT leakage. Windowing the time-domain finite set of samples with a proper function helps to reduce the leakage.

5.1.1 DFT Leakage

By taking only the finite-time interval from the original signal signal, we created discontinuities at the beginning of the taken part and its end. Even if the signal contains only one single frequency, those abrupt discontinuities will tend to broaden the spectrum component. The phenomenon is called the spectral leakage – the energy leaks from one frequency bin to the neighbouring frequency bins. Hence, the inconvenience of the leakage is that if other smaller frequency components are present in your signal, they may be covered in the spectra by the leakage.

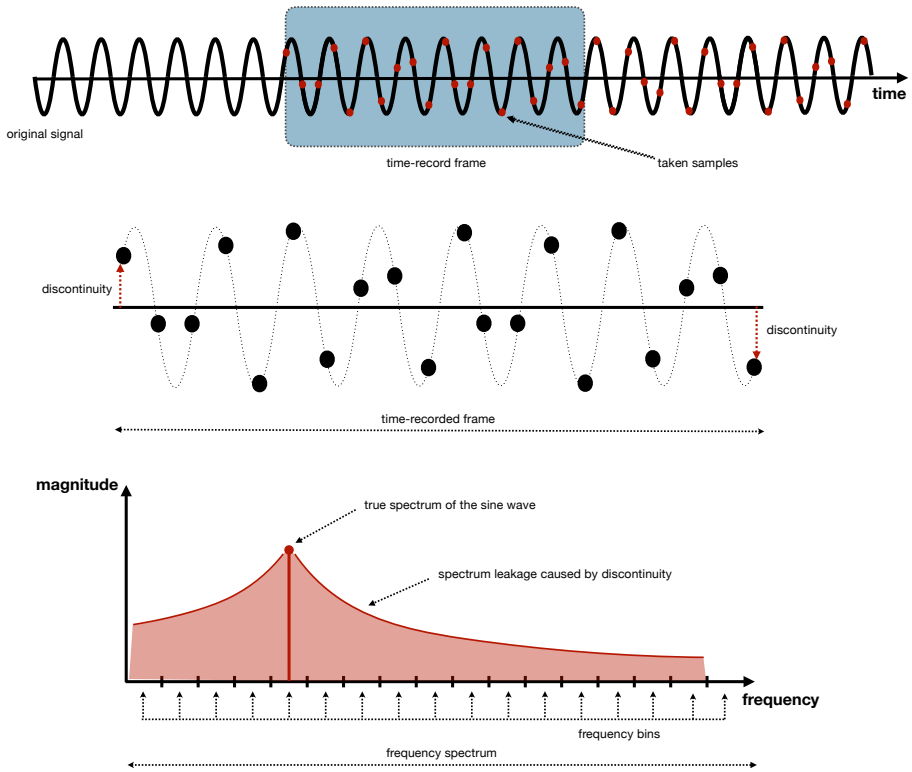


Figure 5.1 – Illustration of windowed signal by taking a finite set of time samples resulting in spectral leakage.

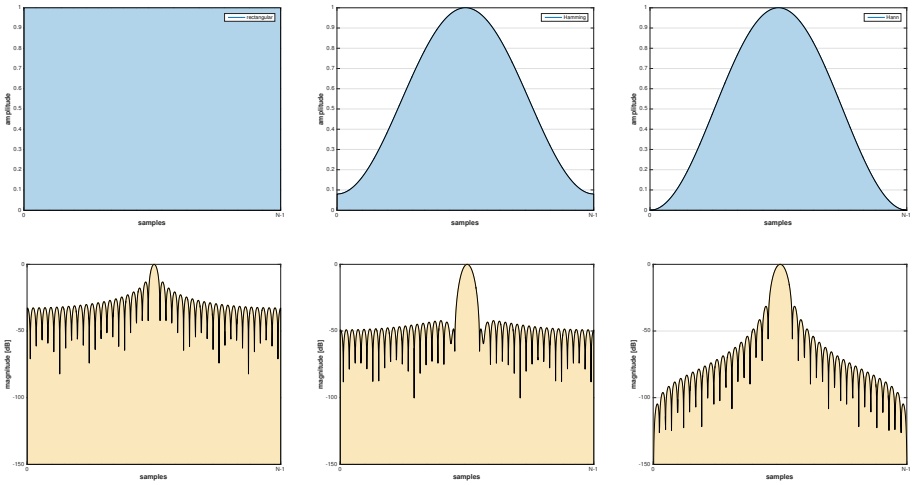


Figure 5.2 – From left to right: rectangular window, Hamming window, Hanning window. Note that edge values of the time domain signal are zero-forced fully with Hanning window. That leads to the disadvantage of the windowing – if the edging samples contained a signal information, the information is lost.

The useful method to suppress the leakage is to weight the STFT window. Window weighting is a change of the window shape, from rectangular to a different one. The windowing functions are typically shaped as a symmetric gradual function with the maximum in the middle of the window and two minims at the edges of the windowing function. In the next figures, you can find several common windowing functions 5.2 [1]. The spectrum magnitudes of windows are plotted in the logarithmic scale. Notify the differences between the sidelobe minimization and the width of the main lobe representing the frequency channel. It can be seen that each of the function gives different results. That provides us freedom in choosing the suitable one for our case. The largest leakage is caused without any weighting – the rectangular window. On the other hand, this window has the narrowest mainlobe. Narrowing the shape of the window, the lower the leakage is and the wider the mainlobe becomes. More on this topic can be found in [64].

5.2 DFT As a Filter Bank

The DFT can be perceived as an filter bank. It is worth noticing the matched principle of such a filter bank. Considering that we sample the time-domain with an constant time step, the center frequency of every bin is then spaced equidistantly. The width of each filter is defined by the size of the taken DFT. Since the response of each frequency bin – the filter from filter banks – is not

[1] Of course, there is a large variety of many other windowing functions like Blackman or Kaiser window, etc.

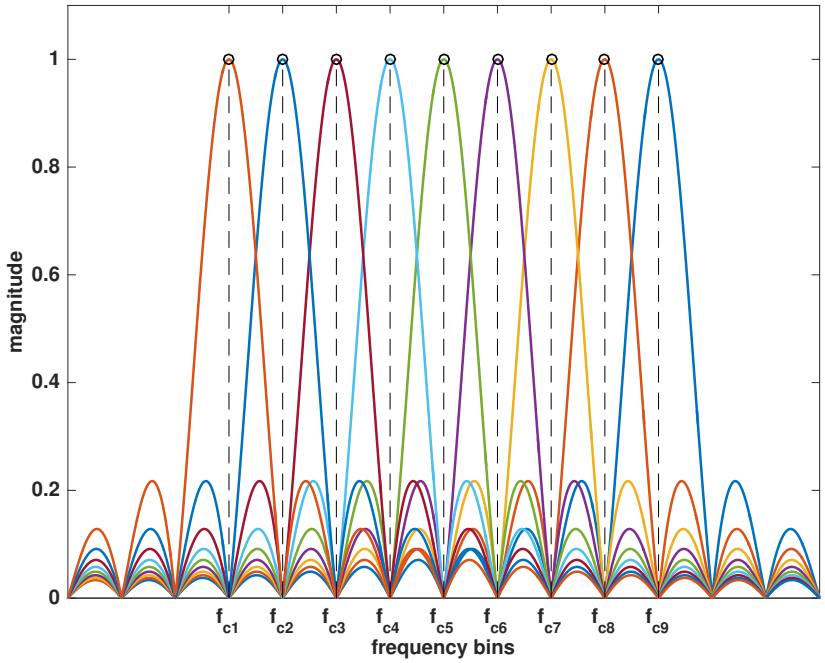


Figure 5.3 – The realistic frequency characteristic of the DFT filter bank. The centers of each bin are marked as $f_{c1}...f_{c9}$. The location of the centers corresponds to the resolution of the taken DFT: $f_{ck} = (k-1) \cdot \frac{f_s}{N} + \frac{f_s}{2N}$ where k = 1 ... 9 for our figure.

flat, it results in scalloping loss. If the signal frequency does not match the center frequency of the bin, the magnitude will suffer from scalloping loss. Moreover, in the picture 5.3 we see the sidelobes of each frequency bin overlapping with neighbouring frequency bins. The minima of the sidelobes are exactly at the centres of the frequency bins. If the frequency of the signal is not matched to the bin center, leakage in the spectrum will appear.

Since we have no clue about the frequency channels (bins) across an unknown FHSS signal is hopping, we can hardly match the DFT filterbank used in the STFT analysis properly. The potential loss of our STFT analysis is the ratio between the best case – center bin matched, and the worst case – halfway between bin centers. Referencing from [65] the value may be up to -4 dB. The scalloping effect can be mitigated by a larger DFT as with an unknown signal, we have higher chance to hit the center of the frequency bin if we have more of them.

5.2.1 Zero-padding

To relieve the effect of scalloping loss and thus improve the estimation of the spectra, a useful technique is adding zero-valued samples at the end of the original set of the samples collected during the short-time interval. With the zero-padding, we will not obtain a finer frequency resolution, i.e., we will not be able to recognize two closely-spaced frequencies. It only allows us to better localize the true magnitude peak in the spectrum, in other words, mitigate the

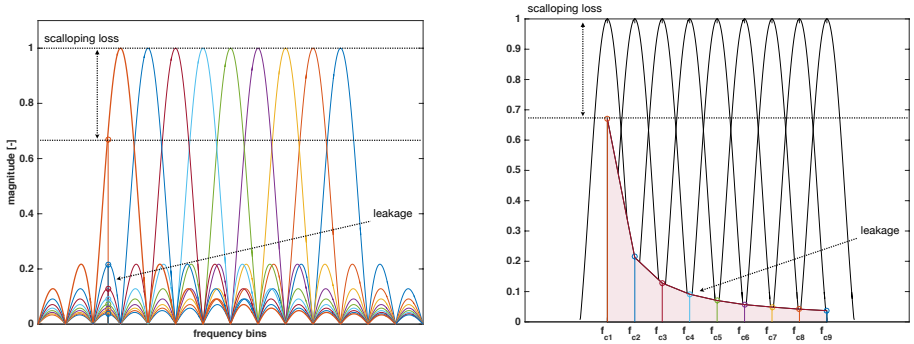


Figure 5.4 – Non-centered frequency, scalloping loss and leakage.

scalloping loss. In some cases, the zero-padding can be helpful. The advantage is we do not need to wait for more true samples; we just add zeros. Nevertheless, then the DFT must be computed from more samples, which means more calculations. In the end, this increase the processing time. In this project, we do not use the zero-padding, however, it may be considered as a future improvement.

5.2.2 Processing Gain

We now exploit the principle of the DFT filterbank. With a DFT computed from a larger number of samples, we reduce the Dirichlet kernel width; that means the bandpass filter from the DFT filter bank is narrowed[2]. The motivation can be found in the figure 5.5.

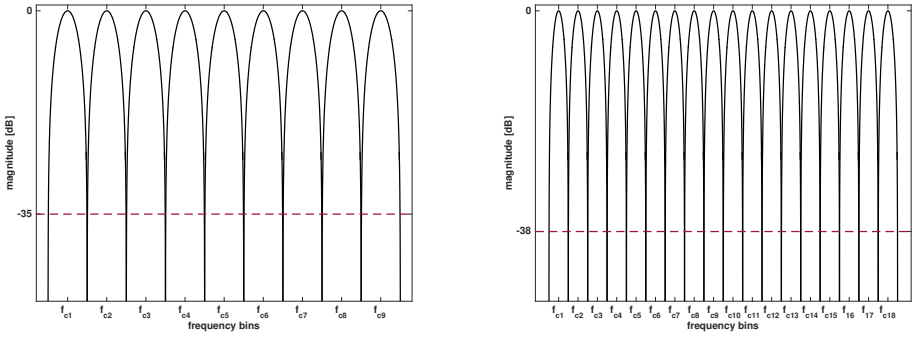


Figure 5.5 – Comparison of two differetnly large DFT filter banks.

We see that the floor of the background noise decreases with the narrowing frequency bin, thus the SNR increases.

[2]The situation is similar to the analog spectrum analyzer, where the resolution bandwidth filter represents the bandpass filter.

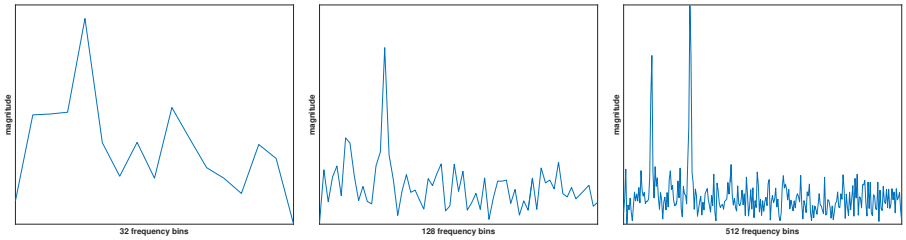


Figure 5.6 – The way how to find a signal buried in noise by narrowing the filter bandwidth, in other words increasing the number of the samples, from which the DFT is computed.

The following pictures in **??** refers to the practical examples where it is clearly visible how the increased number of points of DFT can be helpful to identify the frequency peak in the presence of background noise. Trying to quantify the improvement with a computation of larger DFT, we can formulate the expression:

$$SNR_{2N} = SNR_N + 10\log_{10}\left(\frac{2N}{N}\right) \tag{5.4}$$

Where N is the number of samples used for DFT computation. Taken from [65], equation 3.33

That gives us 3dB SNR increase if the number of samples is doubled. The cost is the computational complexity of a larger DFT.

5.2.3 Integration Gain

Another technique enhancing the estimate of the spectra computed by the DFT is an integration of the individual computed spectra results over time. Presuming the AWGN noise received in the frequency bins, we can reduce the AWGN noise variance by averaging multiple previous bins as it is showed in the figure below 5.7.

Two class of integration are defined. In case a receiver is synchronized to the received signal, coherent averaging can be utilized. Nevertheless, synchronization is often difficult as with the unknown drone FHSS signal. Then, non-coherent averaging can be used.

The non-coherent averaging practically means that the received frequencies do not hit the center of our frequency bins. That provides slightly worse gain performance comparing to the coherent averaging, however, it still reduces the variance of the background noise power significantly. The improvement can be expressed as:

$$SNR_{incoh-gain} = 10\log_{10}\left(\sqrt{N}\right) \tag{5.5}$$

SNR improvement in decibels, where N is the total number of averaged DFT spectrum frames. Equation taken from [65] as 11.17.

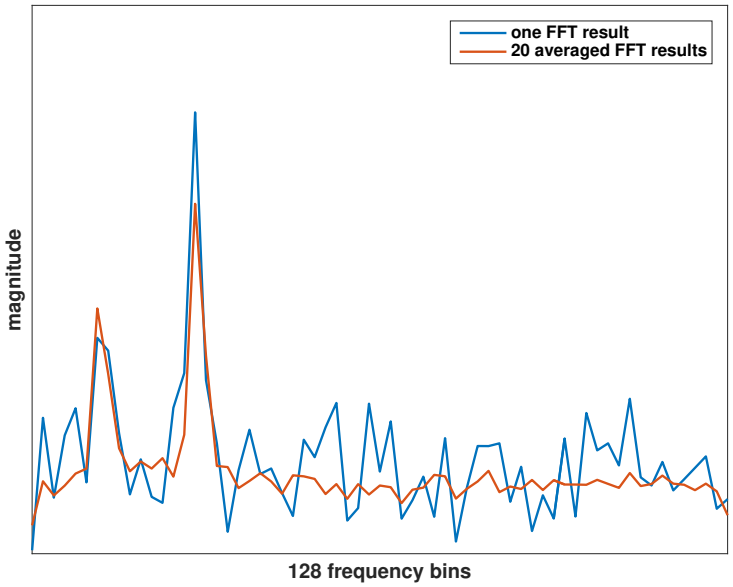


Figure 5.7 – The realistic frequency characteristic of the DFT filter bank.

In terms of one spectrum and multiple averaged spectra and their variances, we can see that the improvement is proportional to the number of total spectra used for averaging.

$$\frac{\sigma^2_{kFFTs}}{\sigma^2_{singleFFT}} = \frac{1}{k} \tag{5.6}$$

σ^2_{kFFTs} is a noise variance of k averaged DFT frames. $\sigma^2_{singleFFT}$ is a variance of one DFT spectrum frame only. Equation taken from [65] as 11.19.

5.3 Fast Fourier Transform

One of the most appealing advantage of STFT is that we can exploit the computationally efficient algorithm called fast Fourier transform, which can considerably decrease the time necessary to compute the DFT from the STFT windowed signal. The complexity of the original DFT algorithm is $O(N^2)$, meanwhile the FFT algorithm reduces the complexity to the $O(NlogN)$. The efficient FFT algorithm is therefore suitable for FPGA applications, where the resources are limited, and we will benefit from the FFT algorithm in our work. If a reader is interested in details, they can be found in [66], [67], [68], [65].

5.4 Chapter 5 Summary

On the past lines we explained the properties of the STFT analysis tool we are using in our project. As we could see, there are many aspects we need to be

aware of before using the STFT. We learned that with very simple techniques, e.g, the integration gain, we can enhance the estimate of the spectra. Last but not least, understanding this chapter is necessary to set the STFT analysis in the best way we can.

Chapter 6

Measuring an unknown FHSS signal

The following chapter describes the process of measurement of FHSS signals. The measurements are recorded with the software defined radio USRP 2953r, which parameters are stated in the chapter 3. The record is then processed and analysed in the MATLAB environment. First, we record a measurement of conventional Bluetooth device – the wireless speaker which is available on the market. We will be seeking a match with Bluetooth standardized characteristics listed in the standard [21]. Second record contains an FHSS signal coming from a drone controller. No a priori knowledge of the signal is known. We can rely only on the recorded measurement, referenced literature and assumptions we have made.

6.1 Setting the USRP

To record the signal, we need to configure the USRP receiver properly. To do that, we use the *USRP RIO 120 - 160 MHz Single Streaming* project[1] available in the LabVIEW environment[2]. This *Streaming* project provides an elementary setting of the radio receiver directly from the host PC. It sets the received signal bandwidth, its center frequency, and other parameters.

The *Streaming* project can be split into two main parts – the FPGA part and the host PC. The FPGA part process the data bits coming from the ADC. Since the speed of the ADC is fixed to 100 Msps[3], inside the FPGA, the signal needs to be decimated to the user-defined sample rate, which corresponds to the received signal bandwidth. Recall several facts. First, the signal has been split into I and Q components. Then, as the ADC inside the USRP has resolution of 14 bits, which gives us 28 bits assuming I and Q component together, the data bits are padded and cast to the unsigned 32 bit type (U32). The data in U32 format are

[1] in the following text stated only as the *Streaming* project

[2] the full name of the program is LabVIEW Communications System Design Suite 2.0 environment as it was mentioned in the chapter 2

[3] Mega samples per second

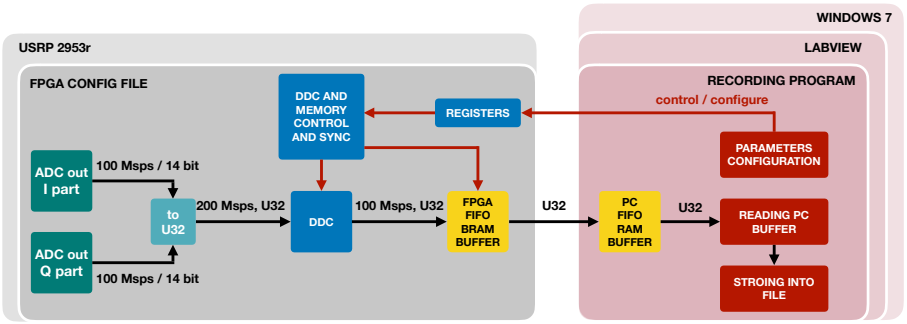


Figure 6.1 – Simplified block scheme of the modified *Streaming* project, which was used to configure the SDR receiver and storing the data during the measurement recording.

forwarded (eventually buffered) to that part of the block RAM memory, where they can be later accessed by the host computer. This whole operation runs inside the FPGA of our USRP device at the 200 MHz clock frequency. To provide high versatility of the SDR's possible configurations, the design includes shift registers, which are accessible from the host PC. Inside those shift registers, the parameters from the PC are being read, thus, it is not necessary to go through the whole procedure[4] of creating a new bitfile envisioned in 3.6 only to change for example the received band center frequency.

The host PC side of the *Streaming* project fetches the data from the block RAM buffer placed in the FPGA. Then the received signal data can be plotted in time domain, or the spectrum of the signal. Those are implicit functions of the *Streaming* project. Nevertheless, we need to realize several facts. The LabVIEW program runs under the Windows 7 operation system. When the user wants to continuously plot the received signal with the bandwidth of tens of MHz inside the LabVIEW [5], or even compute the Fourier transform[6] of such signal, the computer with such operating system is simply not fast enough to perform such tasks. It is not within the scope of this project to describe all the mechanisms behind this process. We just mention that from its nature, the x86 architecture with non real-time operating system, which the Windows 7 is, we cannot obtain reliable results. Hence, outsourcing as many operations as possible inside the FPGA is desired.

To deal with this limitation, we are going to modify the *Streaming* project not to plot the received signal, but only to store the signal time domain data into a file. The recorded data stored in the file will be later loaded in MATLAB, hence

[4] even though the procedure is simplified in the LabVIEW environment

[5] Let's do a simple computation, With the 100 MHz ADC sampling frequency for each I or Q branch, we obtain end up with 32 bits every 10 ns. In other words the data rate is 3.2 Gb/s, which is 400 MB/s. That certainly does not violate the limit of the PCIe x4 interface stated in the table 3.1. However, mathematical computations and graphical interpretation on the screen of the PXIe-8880 PC with Win 7 is simply not feasible in a reliable way.

[6] even using FFT efficient algorithm

parameter	value
frequency range	2.4 – 2.5 GHz
center frequency	2.45 GHz
bandwidth	100 MHz
acquisition mode	continuous
FPGA buffer size	130 000 elements
PC buffer size	100 000 elements

Table 6.1 – USRP configuration parameters set in the LabVIEW environment.

analyzed a posteriori[7]. The data transfer itself is performed via Direct Memory Access (DMA), which consists of a block RAM memory buffer at the FPGA side and a memory buffer at the host computer side. Storing the signal waveform into a file, we are making an assumption that with proper size of those buffers, the LabVIEW program can store the data reliably. We presume that storing process – simple readdressing of the memory cells – is viable on the host PC as not complex calculations are involved[8]. For all measurements we will use the parameters stated in the table 6.1.

6.2 Signal Recording

Before measuring a drone remote controller's FHSS signal, we performed a measurement of a wireless speaker connected to a computer via Bluetooth technology. Bluetooth is well-known for using FHSS signals and it is standardized [21], so we will be able to compare taken measurement with the defined standard.

The measurement was not taken under any special conditions. It was recorded in regular room inside which our software-defined radio testbed was placed, so as the laptop communicating with the Bluetooth speaker. The distance between all the equipment was approximately two meters. With such distance, we anticipate that the received signal will be the strongest signal on the record[9]. Besides the Bluetooth FHSS signal, we expect other signals to appear on the record, e.g., the WIFI signal as the laptop is connected to the wireless network access point via that technology. From a practical point of view, there is no need to perform a measurement with the strict laboratory conditions. The potential detection system must also work under realistic conditions[10]. Because we measure the ISM 2.4 GHz band, other communication technologies might appear on the record. The used equipment is listed in the table 6.2.

[7] there we see the motivation to implement all the signal processing operations inside the FPGA and thus aim at real-time processing scenario

[8] Later, during the measurements, to check if all the data have been stored, we come up with simple and naive technique. We measured the signal recording time. Then, with the signal bandwidth of 100 MHz, the resulting file must be 400 MB multiplied by total recording time.

[9] as the he free-space loss (FSL) will be small

[10] As a proof, that the Bluetooth signal is present, all we need to do is to turn the speaker aloud.

equipment	model	detail
transceiver 1	Lenovo X230	[69]
transceiver 2	SoundCore mini	[70]
recording receiver	USRP 2953r	chapter 3

Table 6.2 – Model specification of the used equipment for the measurement.

6.2.1 Bluetooth Signal STFT Analysis

To analyze the recorded Bluetooth FHSS signal we use Short-Time Fourier transform as we agreed on at the end of the chapter 4. We remind that the analysis of the record is performed in MATLAB. For the analysis, we modified the script we used in chapter 4. To load the recorded data in MATLAB, we reused and modified the script provided by our team colleague David Löschenbrand. Both scripts are listed in the appendix A.2, A.3.

Based on the knowledge from chapter 5, and several experiments with the different STFT parameter settings, we ended up with the configuration as in the table 6.3.

DFT length	512 points
window function	Hanning
overlapping	not implemented
integration gain	not implemented
zero-padding	not implemented
time resolution	5.12 μs
frequency resolution	195.3125 kHz

Table 6.3 – Parameters of the STFT analysis technique.

Since the whole record was about 9 seconds long, which would make the spectorgram plot unreadable, we took a smaller time interval from the record and plotted the STFT analysis results in figures 6.2, 6.3 and 6.4.

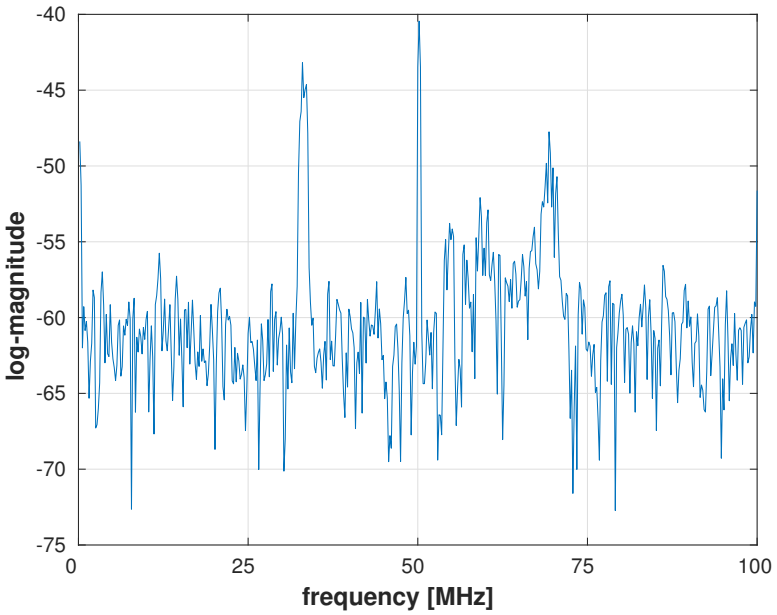


Figure 6.2 – One 512 bin DFT spectrum frame. In the picture, we distinctly see the "monochromatic" peak of the local oscillator and its mirrored image. Further, we see the Bluetooth signal and faded WIFI broadband OFDM signal.

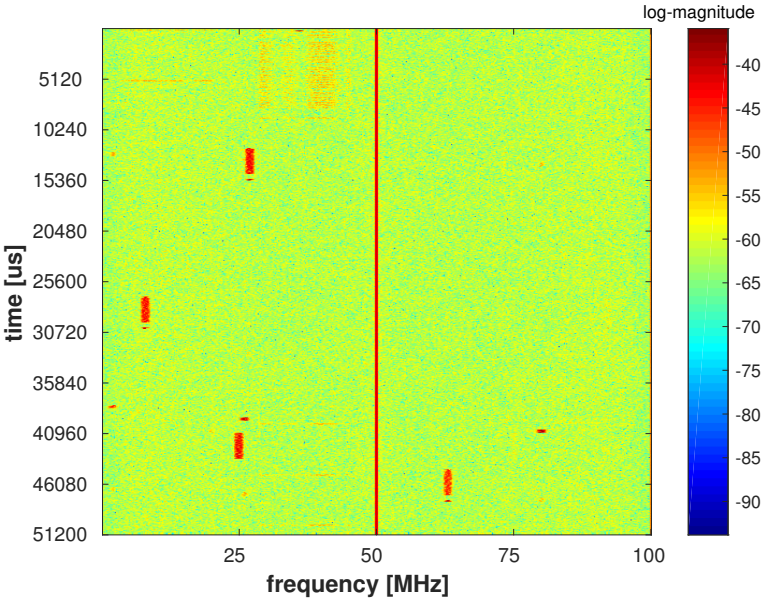


Figure 6.3 – Spectrogram with 10^4 rows of 512 bin DFT spectrum frame. We recognize the local oscillator in the middle of the spectra and its image at the edge. Then, we finally see the hopped FHSS signal used by the Bluetooth. WIFI communication is also nicely visible.

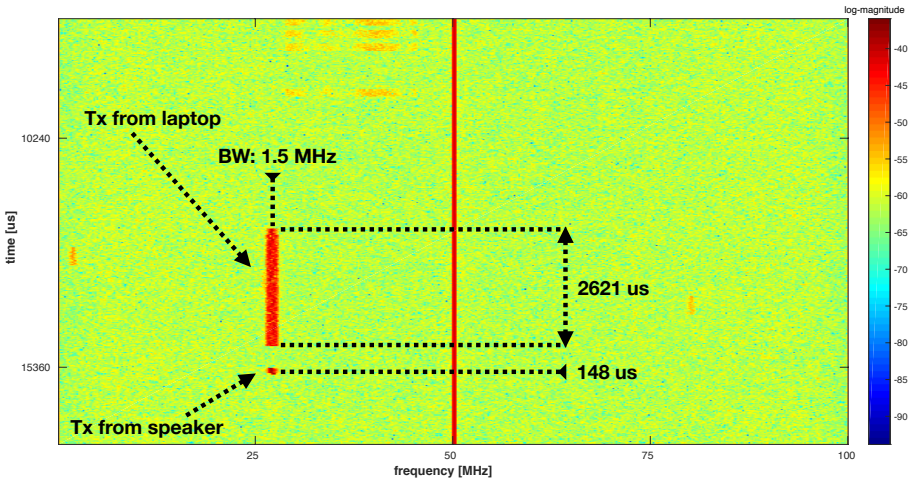


Figure 6.4 – Zoomed spectrogram containing only packet transmitted by the laptop towards the speaker and "ack" packet sent by speaker back to the laptop.

When we zoomed in the previous figure 6.3, we could manually[11] estimate the parameters of the FHSS signal – instantaneous channel frequency and hop duration. Estimating the bandwidth of the instantaneous channel frequency, we need to remember that we used the Hanning windowing function, which broadens the mainlobe of the Dirichlet kernel function as explained in chapter 5. Thus we might consider scaling the estimated channel bandwidth with respect to the selected windowing function.
From the figures above, we see that STFT can be a powerful tool to solve our sensing and FHSS signal parameter estimation problem. We can now move to the analysis of the unknown drone controller signal.

6.2.2 DJI Drone Controller Signal Recording

For the recording the drone controller's signal, the settings of our SDR receiver remained the same as for the measuring the Bluetooth speaker signal, viz. table 6.1. The equipment for the measurement is listed in the table

equipment	model	detail
transceiver	DJI 6 drone controller	–
recording receiver	USRP 2953r	chapter 3

Table 6.4 – Model specification of the used equipment for the drone controller signal measurement.

6.2.3 DJI Drone Controller Signal STFT Analysis

For the analysis of the recorded data, we used the same script as for the recorded Bluetooth signal data 6.3. Again, the record is too long to plot it all, hence we

[11] by inserting a markers in MATLAB

are plotting only a smaller time interval from the record. Restults are shown in the figures 6.5, 6.6 and the zoomed detail in 6.7.

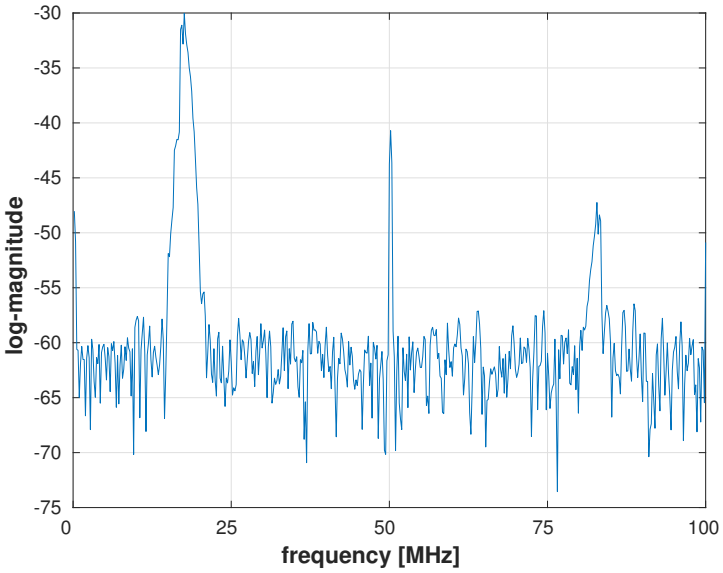


Figure 6.5 – One 512 bin DFT spectrum frame. In the picture, we distinctly see the "monochromatic" peak of the local oscillator and its mirrored image. Further we see narrowband drone controller's signal and its image.

In the figure 6.5, we can see one time snapshot represented by 512 large DFT spectrum frame. We clearly see the local oscillator peak in the middle and its mirrored image at the left edge of the spectrum. Then, two narrowband signals are visible. In fact, due to the insufficient[12] image rejection of our receiver, we see only one signal and its mirrored image. The left and stronger peaks is the true drone's controller FHSS signal. From this spectrum plot, we are able to estimate the channel frequency of the hop and its bandwidth (assuming 3dB decrease).

The hop duration can be estimated from the spectrogram in the figure 6.6. Looking at the picture, we see the hops of the narrowband FHSS drone controller's signal, its mirrored images and also the WIFI signal, which was present as we took the measurements inside the office, where the signal from an AP[13] placed in the neighbouring hallway could be received.

[12] We need to realize that the drone controller transmits the signal with 20 dBm output power and we placed the controller only a few meters far from the receiver. Thanks to low FSL, the received power is stronger than it would be in more realistic scenario.

[13] AP = access point

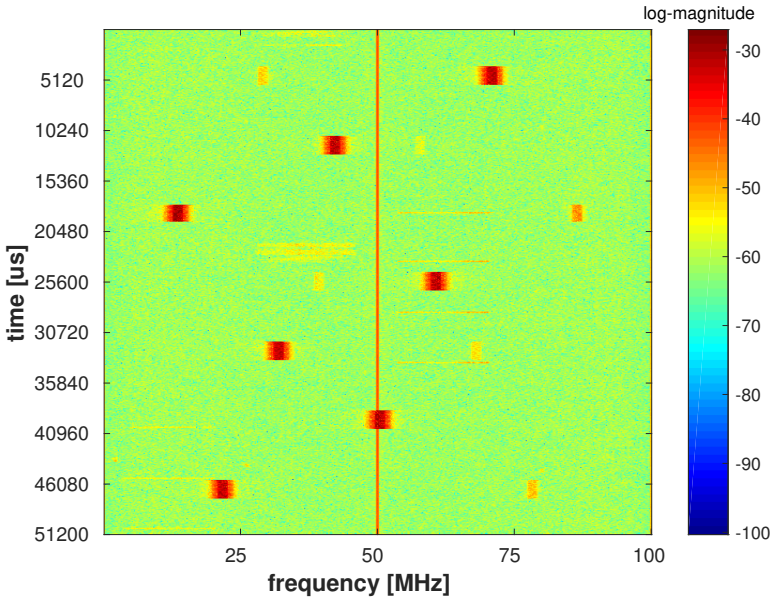


Figure 6.6 – Spectrogram with 10^4 rows of 512 bin DFT spectrum frame containing the hops of the drone controller FHSS signal.

Zooming the detail of the previous figure 6.6, we were able to estimate the hop duration of the drone FHSS signal. The approach was the same as we used for the Bluetooth signal in the previous section.

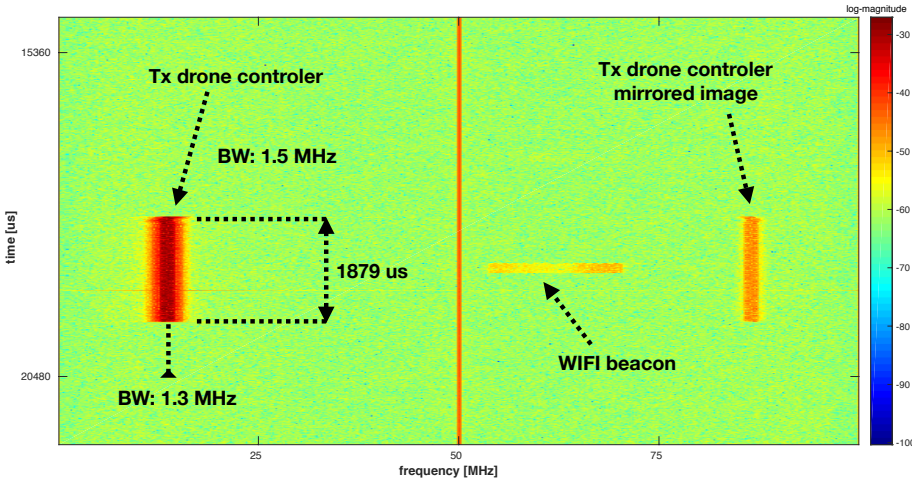


Figure 6.7 – Zoomed spectrogram containing only one hop of the drone controller's FHSS signal.

6.3 Measurement Summary

Measuring the standardized Bluetooth signal, we verified that the STFT analysis technique can be used for estimation of fundamental parameters of the FHSS radio communication link. Looking at the final plots of the Bluetooth signal, an experienced reader may notice several facts. According to the Bluetooth standard [21], the duration of the hop (time-slot) is 625 us. However, as it is also defined in [21], if the radio environment conditions allows, the hop duration can be extended up to 5 time-slots – 3125 us. The value of the hop duration we estimated corresponds to such scenario. We assume that this was possible due to the close placement of the laptop and the Bluetooth speaker, thus low FSL and therefore high SNR. We also observed that the two consecutive master-slave timeslots occupy the same frequency channel.

Then, we took a measurement of the signal coming from the DJI6 transmitter, which is supposed to control DJI PHANTOM 1 drone. We estimated the parameters of this radio signal exactly as we did for the Bluetooth signal. We conclude that the STFT technique can be used for the purpose of FHSS signal parameter estimation based on the energy detection. Nevertheless, if we are interested in a real-time stand-alone drone signal detection system, we surely cannot read the estimated values from the spectrogram plot. We need to detect the signal in a boolean manner, i.e., the signal is present or not. To propose a solution to such a task, we follow to the next chapter.

Chapter 7

Signal detection

Using the STFT in the previous chapter, we made a first step towards the FHSS signal parameter estimation – we are able to compute a decent spectra covering the whole signal bandwidth, i.e., we can estimated an energy level at a certain frequency. The next step is to determine if the FHSS signal at the certain frequency is present or not. That involves an introduction to basics of detection theory. As a radio engineers, we got the inspiration in the radar detection technology, where we discovered techniques which we found appealing to solve our problem, specifically the Constant False Alarm Ratio (CFAR) technique. This technique help us to determine an adaptive threshold for our STFT spectrogram. Hence, the CFAR algorithm determines the presence of the FHSS drone controller's signal.

7.1 Detection Theory Basic Concept

Detection theory focuses on mechanisms to answer the simple hypothesis test. In our project, that hypothesis can be formulated as:

$$\begin{cases} H_0 : S_r[\omega] = n_r[\omega] \\ H_1 : S_r[\omega] = x_r[\omega] + n_r[\omega] \end{cases} \tag{7.1}$$

where the H_0 denotes the scenario, when the received spectrum bin $S_r[\omega]$ contains only noise $n_r[\omega]$, and H_1 corresponds to the situation, when the signal $x_r[\omega]$ we want to detect and the noise $n_r[\omega]$ is present. The ideal detection process would be the one, which would make a positive decision always and only when the hypothesis H_1 is true. Unfortunately, that is not realistic for many reasons. The noisy background may fluctuate randomly[1] due to interference and a behaviour of the radio channel. Another reason is that in a realistic scenario, the received signal strength and SNR is not constant, mainly because the radio channel environment again.

[1] At this moment we do not think about the increase/decrease of the noise thanks to processing gain5 or changed signal bandwidth. We still assume the FFT settings as stated in table 6.1.

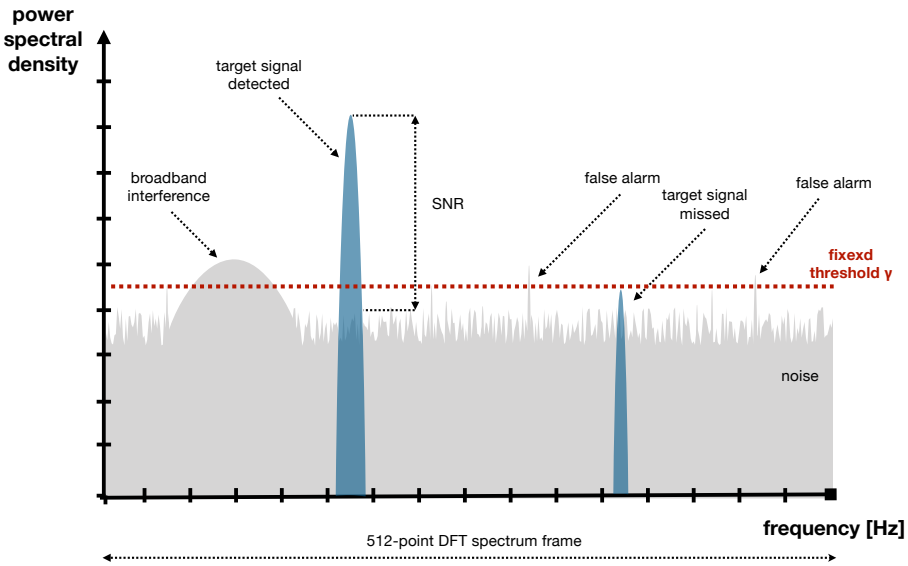


Figure 7.1 – Illustration of detection scenario.

Due to those facts, we cannot resolve the two hypothesis from 7.1 with 100% success rate. As showed in figure 7.1 with a fixed threshold, there is a certain probability, in which the H_1 is being decided by the detector even the correct response should be since only noise n_r is present H_0 – *Type I error*. The reverse case is that the detector erroneously decides H_1, but the H_0 is true –*Type II error*. That gives us four potential detection scenarios depicted in the figure 7.2.

decision	signal	
	present	absent
yes	corect detection	false alaram
no	miss	correct rejection

Figure 7.2 – Possible decision scenarios.

The four cases from the figure 7.2 can be expressed as probability functions:

P_D : Probability of Detection we choose H_1 when H_1 is true

P_{FA} : Probability of False Alarm we choose H_1 when H_0 is true, *Type I error*

P_{MD} : Probability of Missed Detection we choose H_0 when H_1 is true *Type II error*

P_R : Probability of Rejection we choose H_0 when H_0 is true

Then, objective of an optimum detector is to maximize the P_D and simultaneously reduce the P_{FA} and P_{MD}. However, in common realistic situations, those objectives are in contradiction [71], thus a choice between them must be made. In radio signal detection, the problem is typically formulated as maximizing P_D subject to P_{FA} no greater than specified value. This formulation is then typically an objective of Neyman-Pearson criterion leading to likelihood ratio test [72]. With given probability density functions of P_{FA} and P_D, the necessary SNR can be computed [71]. We note that with the increasing target signal SNR and fixed threshold maintaining constant P_{FA}, the P_D increases. Plotting the results of the computations, we obtain a Receiver Operating Characteristics (ROC) graph, characterizing sensitivity of the detector and its false alarm rate.

7.2 Constant False Alarm Ratio Detector

We understand that selecting a value for a fixed threshold is burdensome due to variant behaviour of the background noise and the receiver signal strength. Hence we propose an Constant false alarm ratio algorithm, which is well-known in RADAR receivers, to establish an adaptive threshold for time-domain RADAR pulses. Those pulses are short in time [73], [74], [75], and they look remarkably similar to the FHSS signals we observed in chapter 6 in the frequency domain. Therefore we believe that we can reuse the idea of the CFAR algorithm to obtain a adapted threshold for our frequency spectrum frame. As the name refers, the CFAR algorithms allow us to keep the P_{FA} constant, hence not to overwhelm the boolean outcome by interference and noise.

With the CFAR, there is one more advantage we can exploit. The RADAR pulse detectors needs to mitigate echos – so-called *clutters* – which are caused by the unwanted reflection of the RADAR pulse from ground, sea, rain, or atmospheric disturbances. Looking into [73], [74], we realize similarity between the time-domain clutter and the broadband signal in the frequency domain. Remember, drone's controllers often operates in the ISM bands as discussed in chapter 2, hence we can expect boradband WIFI OFDM signals. Such scenario was also observed in chapter 6. Thus, an adaptive threshold mitigating the WIFI interference would be beneficial.

7.2.1 CFAR Principle

The general idea behind all CFAR processors is computing simple statistics of a reference window, where the reference window is a certain interval taken from the original sequence. In a RADAR detector, the reference window can be described as a tapped delay line of the incoming values from the energy detector. In our case, where we estimating a threshold in the frequency domain, the original sequence is one DFT frame obtained from time-frequency analysis. Therefore, that reference window is sliding over the whole DFT frame in a iterative manner. To provide a meaningful threshold for each frequency bin of the DFT frame, the reference window introduces a structure with following:

CUT: *cell under test* – in the current state of the reference window, for this one bin only the threshold is being computed

neighbouring window: it can be split into two halves – left and right – and it is consisted of number of bins, neighbouring the CUT, from which the data for the statistical computation are taken.

guarding window: it can be split into left and right half; consisted of the very neighbouring bins of the CUT; the bins in guarding window are *not* considered for computation

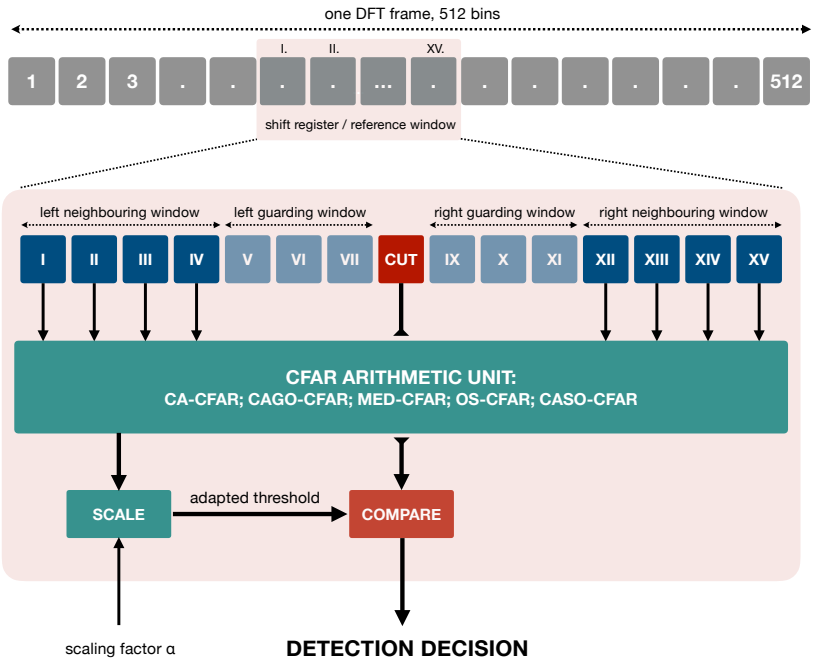


Figure 7.3 – General CFAR processor structure. A reference window of 15 frequency bins is taken from the DFT frame vector. The size of the neighbouring window is 4+4, the size of the guarding window is 3+3. Mind the arabic numerals for bins in original DFT vector and the roman numerals for bins in reference window. We understand that when target signal appears inside the frequency bin under test (CUT), it will not affect the computed threshold. This results in lower threshold for narrow signals occupying only a few bins (fraction of the total neighbouring window).

As the reference window is sliding over the whole DFT frame vector, in each iteration, the data from neighbouring window are used to compute a simple statistics, which is performed in the CFAR arithmetic unit[2]. The computed adapted threshold is then scaled by a factor α. The scaling factor ensures that the resulting threshold is raised over the level of the background original values of the DFT frame vector. In the end, the original value of the CUT is compared with the adapted threshold, which results in boolean solution if the

[2]There is plenty of different options to evaluate the data from the neighbouring window like *Cell Average* - CFAR or *Order Statistics* - CFAR.

signal is present or absent within the CUT – the tested frequency bin $S_r[\omega]$. The structure of the general CFAR processor can be seen in figure 7.3.
We see that the CFAR processor relies on the assumption that the cells around the CUT contain a reliable estimate of the noise in the tested cell – the CUT. A bright reader may notice a problem with the finite length of the DFT vector. In other words, computing threshold for frequency bins at the beginning and at the end of the vector. This issue we solve by assuming the DFT frame as cyclic, hence we wrap the end of the spectra vector to its beginning. The mathematical formulation is given by:

$$\gamma_i = \alpha \sum_{cyc} f\left[u_r\left[\omega\right]\right] \tag{7.2}$$

where f is the CFAR arithmetic function later be used, α is the scaling factor and γ_i is the computed threshold for the ith frequency bin (CUT). The iteration i is done from 1 to number of frequency bins N of the corresponding DFT frame vector.

$$u_r = \sum_{k=i-G-U}^{k=i-G-1} S_{r_k}[\omega] + \sum_{k=i+G+1}^{k=i+G+U} S_{r_k}[\omega] \tag{7.3}$$

where S_{r_k} is the kth frequency bin from the reference window, and where $i \in \langle 1, N \rangle$.

7.2.2 CFAR Arithmetic

Values from the neighbouring window frequency bins can be processed in several ways. The chosen CFAR arithmetic unit can significantly contributes to the final complexity of the whole CFAR processor as well as to the performance of the processor.

Cell Average-CFAR The most simple method is to compute average of all bins inside the neighbouring window:

$$f[u_r] = \frac{1}{2U} \sum_{i=1}^{2U} u_r[\omega] \tag{7.4}$$

where U is the size of one side of the neighbouring window, and u_r is the vector of frequency bins inside the whole neighbouring window.

CA-CFAR processor provides a low complexity of computation, which may be beneficial during implementation.

Cell Average Greatest Of-CFAR Second option is to compute the average of the left and right neighbouring window (viz. 7.3) separately and take the greater value, which is then scaled:

$$f[u_r] = \max\left\{ \left\{ \frac{1}{U} \sum_{i=1}^{U} u_r[\omega] \right\}; \left\{ \frac{1}{U} \sum_{i=U+1}^{2U} u_r[\omega] \right\} \right\} \tag{7.5}$$

According to [73], [74], the CAGO-CFAR technique performs better in presence of sharp-edges clutters. In our case, we can imagine such situation with an interfering broadband signal, faded due to frequency selective channel. The complexity of the CAGO-CFAR can be presumed as almost the same as CFAR, since after two average computations, we compare those two results and the greater is used.

Cell Average Smallest Of-CFAR CASO-CFAR processor is the same as the previous CAGO-CFAR except the lower value of the two compared averages is taken. Having explored the [74], we expect the performance of the CASO-CFAR as the worst.

$$f[u_r] = \min\left\{\left\{\frac{1}{U}\sum_{i=1}^{U} u_r[\omega]\right\};\left\{\frac{1}{U}\sum_{i=U+1}^{2U} u_r[\omega]\right\}\right\} \tag{7.6}$$

Order Statistics-CFAR The last basic method is OS-CFAR. It is based on sorting the values from the neighbouring window u_r in increasing-wise order:

$$u'_r = S_{r(1)} \leq S_{r(1)} \ldots \leq S_{r(2U)} \tag{7.7}$$

where the u'_r is the sorted neighbouring window, in which $S_{r(1)}$ is the smallest value and the $S_{r(2U)}$ is the greatest value. Then one certain value $S_{(rk)}$ from the sequence 7.7 is selected as an estimate of the threshold for the CUT. We see, that selecting the middle value from 7.7, we obtain a median of the neighbouring window.

7.2.3 CFAR Comparison

In the following section we swiftly compare the above described CFAR processors on the test sample spectrum taken from the measured data in chapter 6. The used script is available in the appendix A.4 and A.5.

In the figure 7.3, we recognize the spectrum with local oscillator spikes, FHSS narrow peak of the drone's controller and faded OFDM signal. It is shown that the threshold is adapted to the background noise as it was anticipated. Also, the assumption about the similarity of the clutter and broadband signal was correct since the presence of the OFDM interference is being mitigated by the raised threshold value[3]. The FHSS controller's signal is detected with all of the processors.

Nevertheless, from the picture, we consider that the CASO-CFAR processor (light blue curve) is not very suitable as the threshold is raised not only for the broadband interference but also for the narrow FHSS peak. That may not be reliable in case lower receiver FHSS signal strength. Next, we see the similar

[3]However, we need to keep in mind, that "sharp" interference boundaries, in other words, strongly selective faded broadband signal will result in increase of false alarms, despite using our adaptive threshold algorithm. Although, we believe it is still better case than the fixed threshold.

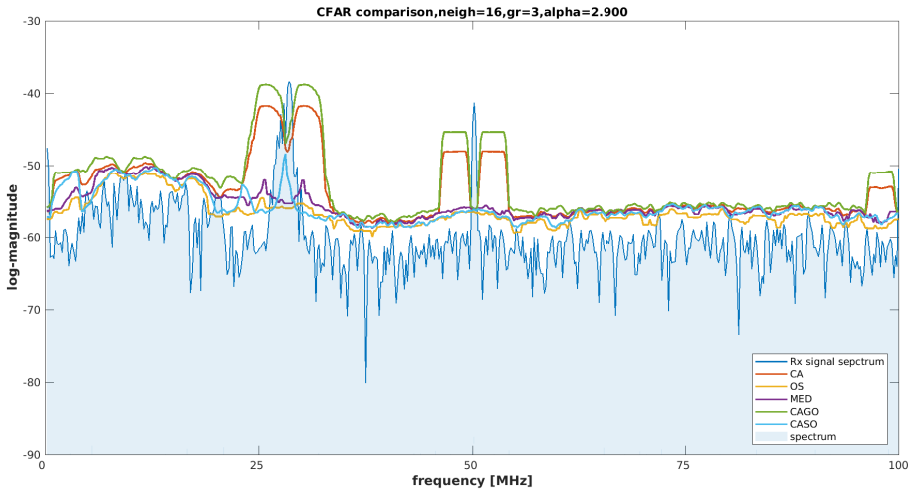


Figure 7.4 – Comparison of several fundamentals CFAR processors. The neighbouring window is 32 bins long (left and right side), the guarding window is 6 bins long (both sides). Scaling factor α is 2.9. For OS-CFAR, the 12th value of the order neighbouring window was taken. The spectrum input of the processors was always in linear scale, after computation, spectrum and threshold were converted to log scale..

performance of the CA-CFAR (red curve) and CAGO-CFAR (green curve) processors. The OS-CFAR (yellow curve) is very effective in following the spectrum "surface" curve and it has also low estimated threshold for the drone peak. However, the selected value from the ordered neighbouring sequence 7.7 might be objective for further optimization. Last, the purple curve corresponds to the OS-CFAR processor with median value selected for the threshold estimation. In this particular scenario, we see it slightly outperforms the 12th order the OS-CFAR processor.

7.3 Results

In this section, for its low complexity[4], we apply the CA-CFAR processor to the recorded data of the unknown drone controller. We reuse the record known from chapter 6, seen in figures 6.5, 6.6 and 6.7. The settings of the time-frequency analysis technique remain the same as stated in table 6.1 – Hanning windowing function of 512 point DFT. From the complex valued bins of DFT frame vector the absolute value is computed. As we referred in chapter 5, we use non-coherent integration to smooth the background noise, thus 10 consecutive DFT frames are integrated together. The detection algorithm is the CA-CFAR processor with 16+16 neighbouring window and 3+3 guarding window. The scaling factor α is 2.3 and the data are processed by the CFAR technique in non-logarithm format[5].

[4] which we are going to exploit in the FPGA implementation phase

[5] Feeding the detection algorithm with linear valued spectrum, we would need a different scaling factor α.

7.3.1 Low SNR Scenario

First, we investigate the performance of our detection system operating on signal with a low SNR. Since we are using the same record as previously, the noise was artificially added in MATLAB environment. We start with the spectrogram of the 512-point STFT compared to the spectrogram obtained after non-coherent integration with period of 10 spectra frames 7.5.

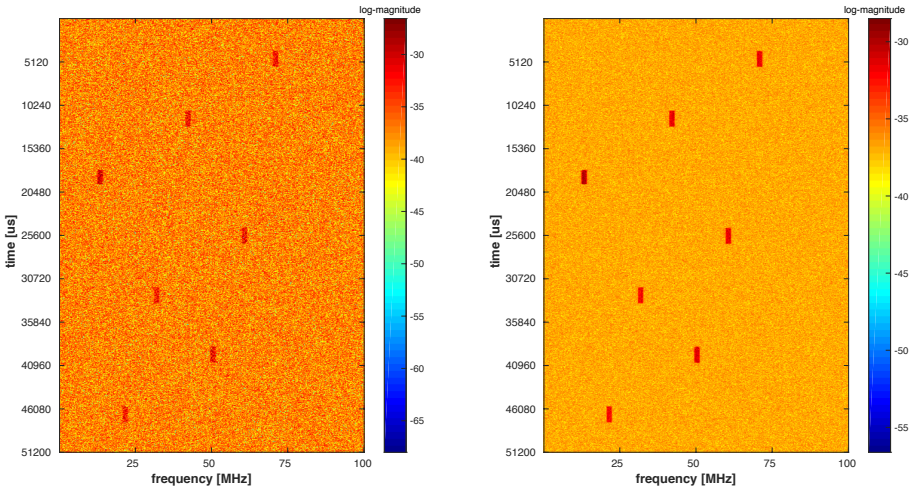


Figure 7.5 – 512-point DFT with Hanning windowing on the left side, right side is the result after non-coherent integration.

By only a visual exploration, it is clear that the integration significantly helps to distinguish the hops of the FHSS drone controller signal from the noise background. Notice that the local oscillator and interference are now buried in the noise, thus, they cannot be seen in those spectrograms.

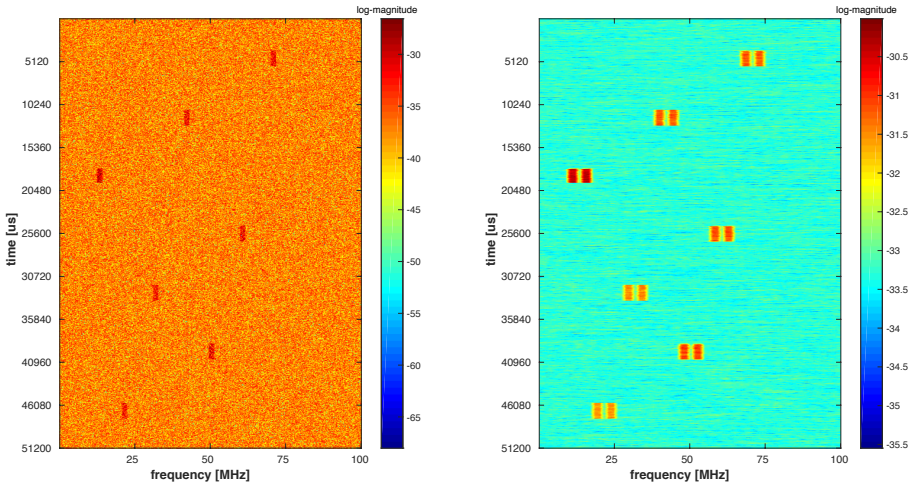


Figure 7.6 – 512-point DFT with Hanning windowing on the left side, right side belongs to the adapted threshold.

The result after non-coherent integration, which helps to flatten the noise floor, is then used to compute the threshold. In the next picture 7.6, we compare the original spectrogram with the spectrogram of already computed adapted threshold. Notice the low threshold value in the middle of each hop, which exactly corresponds to the CA-CFAR curve we have seen in the figure 7.4.

Once the threshold was computed and scaled, we compared the adapted threshold with the original spectra obtained from STFT time-frequency analysis. That gives us boolean decision if the signal is present or not in the figure 7.7.

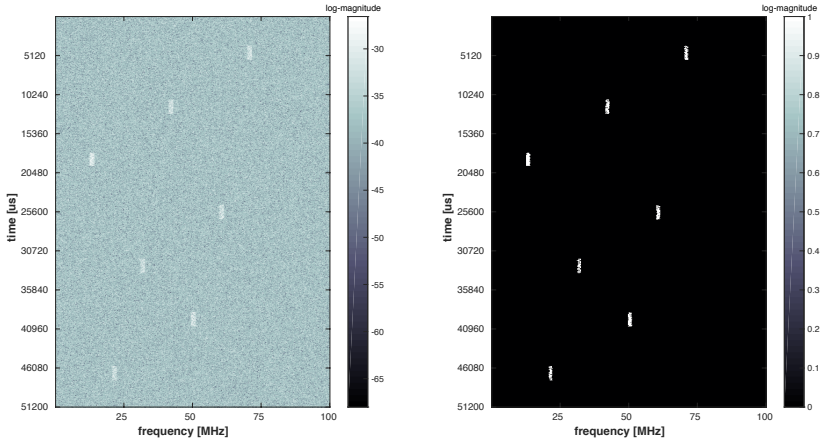


Figure 7.7 – 512-point DFT weighted by Hanning window on the left side, right side is the result after detection.

We now look at the 7.8, where we estimated the hop duration and the instantaneous channel frequency of the hop. The values were read from the markers in MATLAB, thus, an automatic algorithm will be a subject of future work. Nevertheless, author assumes that the most difficult part is to get a boolean (true or false) decision.

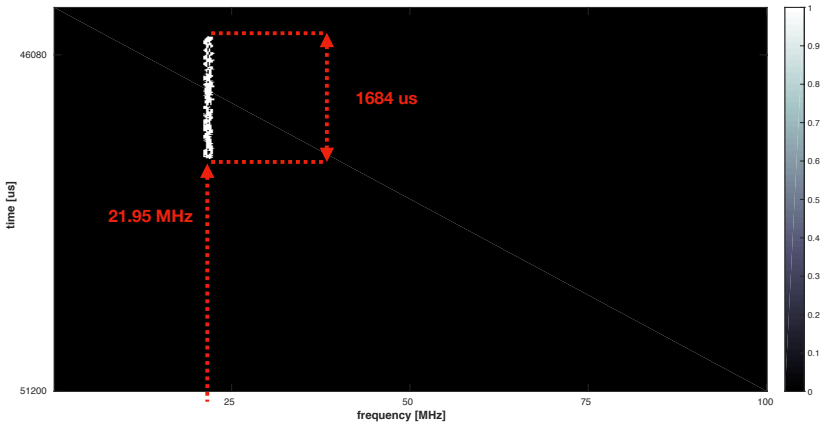


Figure 7.8 – Detail of the detected FHSS UAV radio signal. *The diagonal line is not part of the original figure as it is result of incorrect .eps formating.*

7.3.2 High SNR Scenario

We now move to the original record without adding any noise, hence this section is assumed as high SNR scenario. The settings of STFT analysis and detection algorithm is exactly the same as in low SNR scenario. We emphasize that the scaling factor α remained the same. As showed in the previous section, we now skip plotting the partial processing tasks like non-coherent integration and adapted threshold since the principle and effects are the same for high SNR signal. We move directly to the spectrogram comparing the original STFT analysis plot and the detected hops.

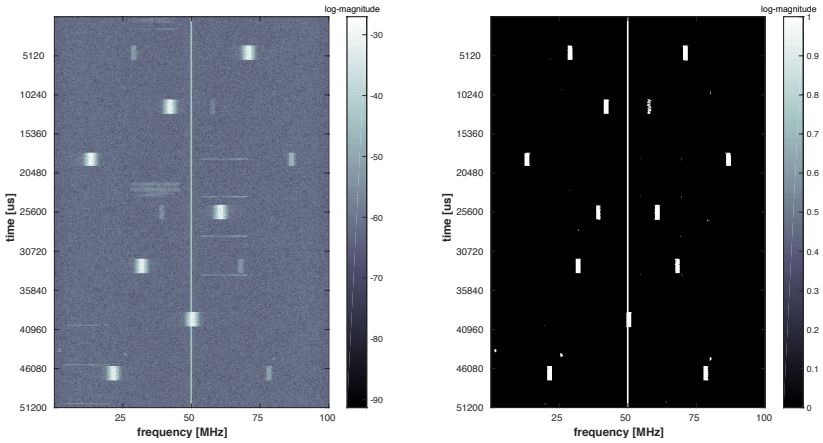


Figure 7.9 – 512-point DFT weighted by Hanning window on the left side, right side is the result after detection for high SNR.

Figure 7.9 shows the desired feature of the CA-CFAR processor. On the left side spectrogram, we see a lot of WIFI interference which was mitigated by our detection algorithm. Naturally, the original record contains local oscillator peaks and images as shown in chapter 6. Due to their high strength, the signal images appear on the detected spectrogram. The very narrow spikes belonging to the local oscillator cannot be filtered by our algorithm, therefore it would be necessary to filter them out in the future improvement.[6]

[6] We did not removed the bins containing the local oscillator peaks on purpose as we believe that the initial demonstration of the SDR capabilities should be shown with all its advantages and disadvantages.

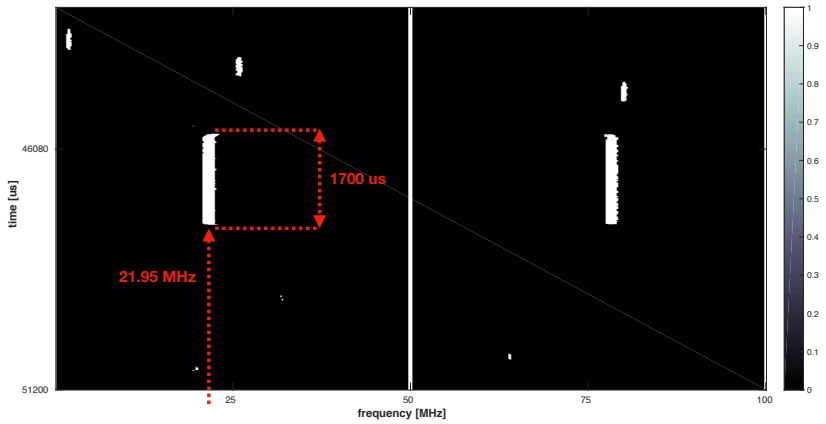


Figure 7.10 – Detail of the detected FHSS UAV radio signal in high SNR case. We observe the false alarmed signals due to their high strength. *The diagonal line is not part of the original figure as it is result of corrupted .eps format.*

In the detailed picture 7.10 we can again see the estimated hop duration and center frequency of the hop. Comparing the figures 7.10 and 7.8 we observe the reason, why we did not estimate also the bandwidth of the channel of use of the hop. Using the CFAR algorithm under a different SNR values will end with different width of the detected signal[7]. Therefore, we do not find an estimation of the channel bandwidth that meaningful.

7.4 Signal Detection Summary

In this chapter, we learned that even with very simple algorithms like STFT and CFAR processors – generally energy detection, we can solve our FHSS signal detection problem. We demonstrated that their use is viable, bringing the advantage of mitigating the broadband interference, which is a common problem due to ubiquitous WIFI technology. We provided approximate estimates of the FHSS drone controller's signal which are necessary for potential jamming scenario. We are aware of the fact, that the DFT integration process (averaging) flattens the noise floor, meanwhile it stretches the hop duration interval, thus it adds a bias[8]. The main benefit of the techniques we used so far is their low complexity, hence we devote the next chapter to implementation of the MATLAB-tested algorithms into the FPGA inside our USRP software-defined radio to aim at real-time signal detection.

[7] However, that is not a big issue for our assumed jamming system, where knowledge of the instantaneous hop center frequency and possibly hop duration is more crucial.

[8] The stretching corresponds to the number of averaged DFT frames.

Chapter 8

FPGA implementation

So far, we described fundamental tools – STFT, CFAR – for a potential FHSS UAV's signal detection system. We verified their potential in MATLAB with the signal measurement we recorded with our USRP. In this chapter, we focus on a implementation of the tested algorithms into the FPGA inside our USRP software-defined radio. Such implementation would allow us to perform an estimation of the FHSS signals' parameters in real-time, which is a crucial aspect for possible reactive jamming.

8.1 Proposed FPGA Program Design

The operations described bellow are the crucial processes performed by the Kintex FPGA. The program design is composed of all the techniques we used in previous chapters. Starting with time domain I/Q samples[1] coming from the ADC of one receiver's channel via the I/O blocks of the FPGA. As we learned in chapter 5, the samples are weighted with a selected windowing function[2]. Windowd samples are filled into a internal buffer. Once we collect all the time domain sample necessary for the STFT, we continue with DFT computation of the windowed. The DFT computation is realized by efficient FFT algorithm. Thanks to FFT computation, the complex frequency domain samples – the bins $f_1 \dots f_N$ – are obtained.

Focusing on energy detection, the magnitude of the frequency bins is obtained by absolute value of the complex number operation[3]. Then, as demonstrated in chapter 7, we average several consecutive DFT frame vectors to flatten the noise floor. Here, we are making an assumption, that one DFT vector represents a very short time interval[4] and that in general, the FHSS signal does not hop that rapidly. Otherwise, the averaging would not be meaningful. After that, averaged result of each DFT vector is fed to the CA-CFAR processor to compute

[1] as learned in chapters 3 and 6

[2] Hanning, Hamming, Blackman, Kaiser, ...

[3] which corresponds to the square-law detection described in radiometer terminology

[4] depending on number of time samples taken for FFT computation, remember that speed of the ADC is fixed, thus size of the FFT has the impact on the time resolution as stated in table 6.1

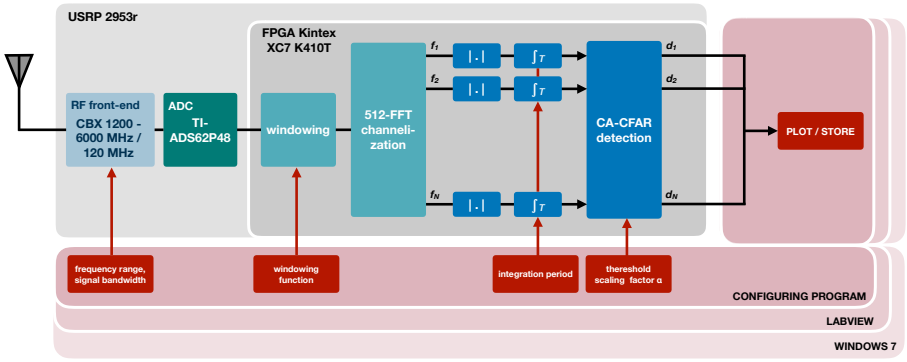


Figure 8.1 – Block scheme of the architecture of the proposed FPGA real-time detection design.

the adaptive threshold, which is then compared with the averaged DFT frame. That gives us binary values $d_1 \dots d_N$ if the energy of the signal has been detected or not – if the signal is present or absent inside the frequency bin. Number of d_N corresponds to the number of frequency bins f_N, thus the size of the taken FFT.

Those values are then forwarded from the FPGA to the designed program running on the host PC, where the detected results can be plotted or stored[5]. To make the design versatile, we implemented several windowing functions, which can be selected from the host PC side. The integration period and threshold scaling factor α is also configurable, as well as the setting of the RF front-end. The size of the FFT is not configurable as such feature could make the fast processing implementation[6] impossible. The options are summarized in the table 8.1. The block scheme of the proposed design is depicted in 8.1.

number of Rx channels	1
frequency range	1.2 – 6 GHz
signal bandwidth	100 MHz
windowing functions	Gaussian, Hamming, Hanning, Parzen, Blackman, Blackman-Harris
FFT size	512 points
integration period	2, 4, 8, 10 or 16 DFT frames
resolution of α	8.8 bits in fixed-point data type
CA-CFAR	16+16 neighbouring window, 3+3 guarding window

Table 8.1 – FPGA design capabilities.

8.2 FPGA Program Design Architecture

Aiming at real-time use, we realize that speed of the processing is essential. To achieve as short processing time as possible, the operations must be performed

[5] We declare that the fetching the data from the FPGA was not fully accomplished in real-time and it remained a task for further improvement.

[6] which can be objective of the future improvement

in parallel. At the same time, each of the operation must be processed at high clock speed[7] and all operations described in the section above must be exactly synchronized.

Briefly look at the picture 8.2. [8] We start with reusing the *Streaming* project used in chapter 6 (figure **??**). We recall that the time domain IQ data are collected with the rate of 100 MHz, which corresponds to the selected signal bandwidth. The time domain samples in fixed point format, where one integer bit is reserved for sign and the rest 15 bits represent the fractional part, from the DDC decimation block are weighted by the windowing function.

8.2.1 Windowing

Windowing of the signal is performed by multiplication of the samples with the windowing function coefficients. For that purpose, the coefficients of Gaussian, Hamming, Hanning, Parzen, Blackman and Blackman-Harris functions are stored into block RAM memory (3.5) inside the FPGA. The multiplication is assumed as an cumbersome arithmetic operation, especially inside an FPGA. Fortunately, Xilinx, the manufacturer of the FPGA inside our SDR, provides ready-to-use DSP48 macro [76]. This macro directly handles dedicated DSP circuit block inside the Kintex FPGA. Use of these DSP blocks is an effective way to perform such arithmetic operations. For the multiplication, one input of the macro has a size of 30 bits and the second one has 18 bits. Before forwarding the data to the multiplier, the data in fixed point representation must be rounded, eventually zero-padded. Then, the DSP macro output results in 48 bits in fixed point type.

8.2.2 Block RAM Buffer

When the time samples pass the windowing block, the samples (black arrows) are written into a BRAM FIFO 1 buffer. To buffer the data, we use a dedicated RAM memory blocks distributed inside the FPGA architecture. Only those BRAMs ensure lossless storage of the data. The memory is a type of FIFO (first in, first out), which is the type of all BRAMs used in our project. To keep the data synchronized, the memory must be mastered by a DRVING LOGIC .

8.2.3 Dirivng Logic

At this point, the DRVING LOGIC 1 block , which defines if the data forwarded to the BRAM FIFO 1 memory are valid and when the data will be read from the BRAM storing the windowed time samples. In principle, the DRVING LOGIC 1 itself is controlled by the DDC synchronization unit. When the data from DDC are valid, then they are windowed and stored into BRAM FIFO 1. All the DRVING LOGIC blocks implement a counter.
When the counter reaches a certain value, a controlling signal is sent to the DRVING LOGIC 2 . The counter increments only if is allowed by its master, in

[7] As understood in figure 3.5, each of the operation can run at different clock speed.
[8] Data signals are envisioned as black arrows, controlling signals ensuring synchronization and setting are depicted in red colour.

this case, when the output of DDC is valid. In this first case, where 512 time samples are needed for FFT computation, when counter reaches the value of 385[9], then the DRVING LOGIC 2 block drives the BRAM FIFO 1 and data can be read. The counting is necessary as the BRAM FIFO 1 is read at a higher clock rate 8.2. That introduces a master-slave hierarchy for each operation we are using in our design and guarantee the synchronization of our design.

8.2.4 Fast Fourier Transform Block

As we know, many multiplication and addition operations are involved in computing DFT, even we are using FFT block. To decrease the total processing time of our detection design to minimum, from now, all the operations are clocked at 400 MHz. Once the DRVING LOGIC 2 2 block allows the BRAM FIFO 1 to be read, FFT block can start computing complex-valued frequency bins.

To compute the FFT, intellectual property (IP) block from Xilinx was used. More information can be learned in [77], where we found how to configure the FFT IP block. The configuration we used is stated in the table 8.2. According to the FFT documentation [77], the FFT block may not be ready for input data all the time. Thus, a signalization to the DRVING LOGIC 2 block must be implemented to keep the FFT block and incoming data from BRAM FIFO 1 memory synchronized.

FFT size	512 points
architecture	pipelined, real-time
input IQ data format	2 x I16 in U32
output IQ data format	unscaled, 2 x 26 bits in 64 bits
butterfly arithmetic	CLB
rounding	convergent
buffers	BRAM

Table 8.2 – FFT block configuration.

8.2.5 Square-law Detection

The channelization into N frequency bins equidistantly covering the signal 100 MHz bandwidth by FFT block is in the form of the complex number. The imaginary and real parts are then squared and added together, in accordance with the equation 8.1 describing complex square-law envelope detector. Mathematically speaking, it denotes the absolute value of the complex number.

$$\sqrt{Re^2[k] + Im^2[k]} \tag{8.1}$$

[9]The origin of the value is form the ratio of the operations. Collecting the time samples is four times slower than the further FFT computation.

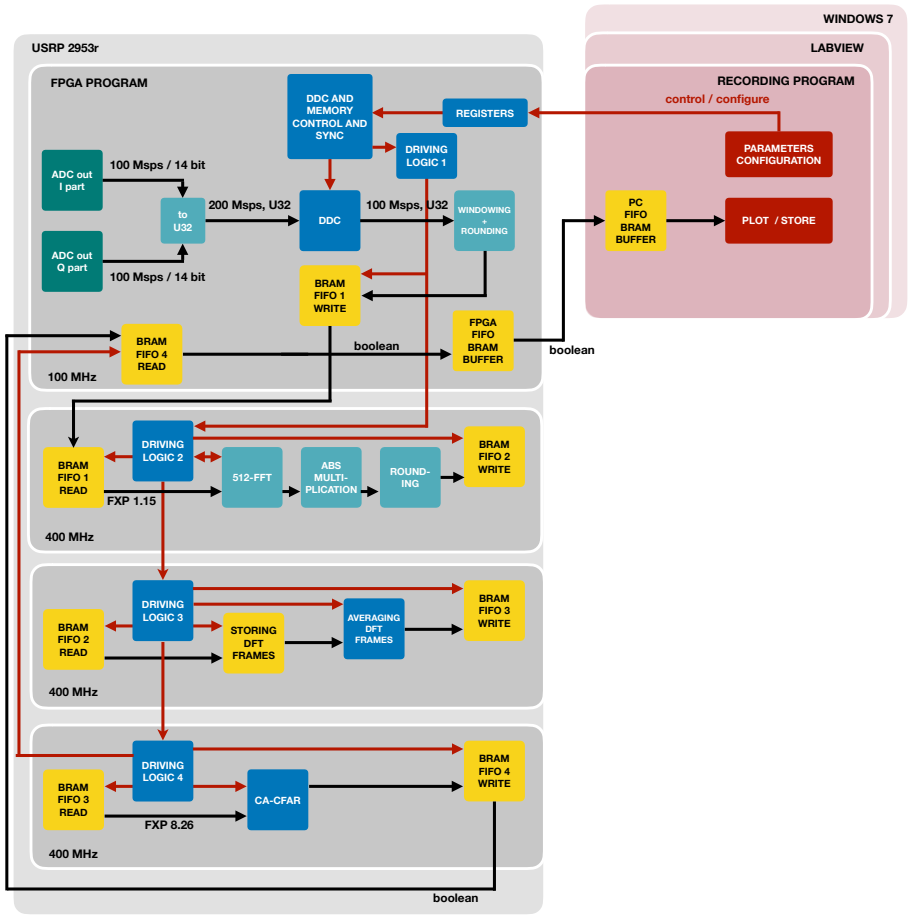


Figure 8.2 – Detailed description of the functional blocks of our program design implemented in the FPGA.

the square root of the result should be taken. Nevertheless, to save FPGA resources and reduce the processing time, we decided not to perform the square root operation. For the multiplication, the DSP macro was used, however, the squaring operation is now performed at 400 MHz. From the behavioral simulation, which is one of the designing tools, we observe that this multiplication takes four clock cycles at our FPGA. This fact needs to be considered for proper synchronization.

8.2.6 Rounding

Similarly to the result of windowing, the output of the squaring operation needs to be rounded to save FPGA resources. Then, the resulting values are written in the second BRAM FIFO 2 buffer.

The DRVING LOGIC 2 2 block then drives the BRAM FIFO 2 to keep the synchronization. That block also drives the the third DRIVING LOGIC 3 block, which allows reading from the BRAM FIFO 2.

8.2.7 DFT Frames Storage

Once the data from BRAM FIFO 2 are being read, we keep 16 consecutive DFT frames stored. This storage can be expressed as a cascaded shift registers, where one register has a size of 512 elements. By writing "cascaded", we mean that the output of the first register is the input for the second register. This principle allows us to read up to last 16 DFT frames.

8.2.8 Averaging DFT Frames

The integration process is basically averaging, which corresponds to the summation of all the outputs of the 16 shift registers. The summation is not computed at once, but follows the pipelining principle. Only two shift registers' outputs are added together at once. The result is then added together with the neighbouring result etc. This saves a FPGA resources and allows the design to run at 400 MHz. On the other hand, this principle introduces a small processing delay.

Making the integrator versatile, a switch is included to select the summation of 2, 4, 8, 10 or 16 consecutive DFT frames. After that, the fractional point of the resulting value is shifted to the left to achieve the division[10]. In case of averaging 10 DFT frames, the fractional point is shifted only by 8 positions. This inaccuracy is then compensated by adjusted ALPHA scaling factor in CA-CFAR algorithm.

Processed are then stored in to the third BRAM FIFO 3 buffer, which is driven by the DRVING LOGIC 3 block to preserve the synchronization. As usual, the driving block 3 controls the reading of that BRAM FIFO 4 buffer in the next 400 MHz section.

8.2.9 CA-CFAR

The CA-CFAR processor design corresponds to the figure 7.3. Its design is based on shift register with the size of 39 elements[11], where the 512 points large DFT frame is continuously shifted. As we described in chapter 7, the problem is the finite length of the DFT frame vector. Thus, the first valid output of the processor – the binary detection result – is the 20th processed frequency bin and this is the first value to be written to the BRAM FIFO 4, which is mastered by the fourth DRIVING LOGIC 4 block. Meanwhile the first 20 frequency bins are stored in the shift register, which is delayed by the $N-(U+G)+1$, where N is the total number of frequency bins (512), U is the size of the one-sided neighbouring window (16), and G is the size of the one-sided guarding window (3). The output of that delayed shift register is then switch as an input of the main CA-CFAR shift register. This warp around allows us to compute the adapted threshold for all frequency bins. The disadvantage is the changed order of the frequency bins, which needs to be considered in later applications. As

[10] Shifting the fractional point is more effective than conventional division arithmetic operation, which is on the contrary very expensive in terms of FPGA resources including processing time.

[11] corresponding to the 16+16

usual, everything is mastered by the DRIVING LOGIC block, which also signalizes to the upper-most master part of the design that the result of detection is available.

8.2.10 Forwarding to The PC

When the boolean result of the detection algorithm is signalized as done, in the 100 MHz, the BRAM FIFO 4 is being read and forwarded to the FPGA-PC transfer buffer. From there, the data are read by the host PC and plotted or stored.

8.3 Processing Flow

To recapitulate the parallel processing of our design, look into the picture 8.3. This diagram flow directly corresponds to the analysis of the signals/vectors inside the FPGA. This analysis can be shown inside the behavioral simulator designing tool. In that simulator, the time axis is represented in clock cycles, so we can check if all the FPGA internal signals representing the values are properly aligned - synchronized. After synthesis and routing our design into the FPGA, the operations with the internal FPGA signals will be performed the same. From the behavioral simulation, we were able to determine the total processing time, which is the time from collecting the first time domain sample from the frame (A) up to the last detected boolean value of the frame (A).

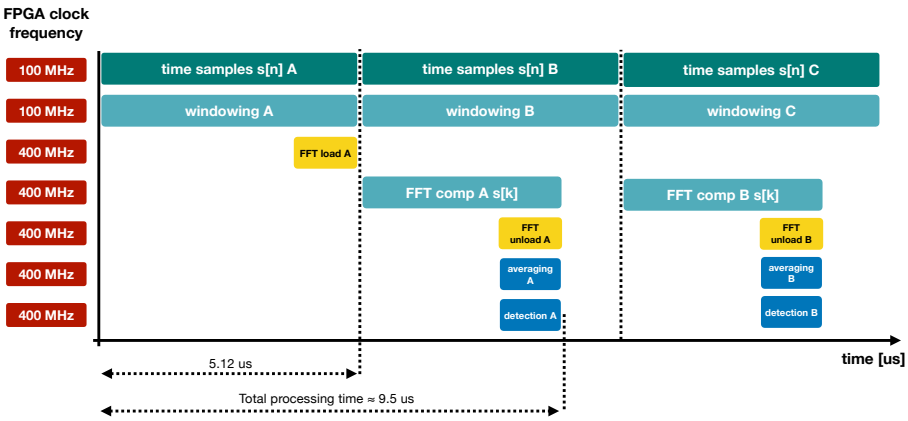


Figure 8.3 – Parallel processing flow illustrating the implementation of our detection algorithms into an FPGA.

Telling the truth, the illustration above is not completely correct. To synthesise and route our design, it was necessary to include additional delays between the operations. Nevertheless, those delays are in a range of a few clock cycles, thus several tens of nanoseconds in maximum. Thanks to all the effort we described, the total processing time is approximately 9.5 us, which we assume as fast enough to encounter the FHSS UAV controller's signals.

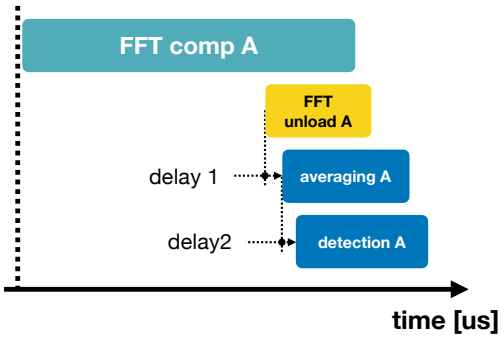


Figure 8.4 – Detail of the processing flow – introducing the inter-operation delays.

8.4 Implementation Summary

The proposed design implements all the features used in chapter 7 to obtain the results in figures 7.5, 7.6, 7.7, 7.8, 7.9 and 7.10. The features are stated in the table 8.2. The correct functionality of the FPGA design was verified in behavioral simulator tool. The generated FPGA configuration bitfile was loaded into the FPGA and the results were plotted on the screen of the PC. Due to not completely solved issue witht the data transfer between the FPGA and PC, the plotted data were corrupted.

To make sure that the FPGA works as it was designed, we created another bitfile, which design was modified. The modification allowed us to send the known time domain samples from host PC, perform the processing inside the FPGA and load them back. The time domain samples were transferred into the FPGA in chunks with defined size of $512x196$ samples. Once the data were loaded into the FPGA, the host PC was waiting till the FPGA processing was done. The processed data were then transferred back to the PC and plotted. This sequential[12] scenario worked impeccably. Hence we claim that our design is correct, only the forwarding data to the host PC sufficiently fast needs to be fixed.

[12] loading the data into FPGA, waiting and unloading from FPGA to be plotted

Chapter 9

Conclusion

The objective of our project was to design a software-defined radio system, which would be able to estimate unknown parameters of an UAV radio communication work. Thanks to the referenced literature, the objective was accomplished by configuring the software-defined radio as a signal recorder, then the record was analyzed in MATLAB environment in which the unknown parameters of an real-world UAV radio controller FHSS modulated signal were analyzed. Parameters like instantaneous center frequency of the hop and the hop duration were successfully estimated.

In chapter 2, we described the current problem we are facing with the expansion of UAV – flying drones. We characterized the FHSS modulated radio link they use. In chapter 3 we described the concept of current software-defined radios. We introduced its general positives and limitations as the equipment we used in our project is based on the SDR architecture. The fourth chapter (4) includes a simulation of the FHSS radio signal and comparison after time-frequency analysis – short-time Fourier transform, wavelet transform and Wigner-Ville distributions – of the simulated signal. Based on the conclusion summarized in the last section of this chapter, we decided to use the STFT in the rest of our project. To understand the important time-analysis tool we used in our work, we dedicated chapter 5 to describe STFT properties.

The chapter 6 is devoted to the real-world measurement of the FHSS signals and their a posteriori analysis. First, our SDR is correctly configured, then measurement of a Bluetooth signal and a drone controller's signal is recorded. the results are then analyzed by the STFT. Chapter 7 explains simple detection algorithms – the CFAR processors. Several CFAR types are then tested on the measured UAV signal record. Based on the basic demonstration, the CA-CFAR processor is selected. In this chapter 7, thanks to the STFT and CA-CFAR detection algorithm, the parameters of the UAV FHSS radio link are estimated for low and high SNR scenario.

After verification that the methods we used in MATLAB are viable, we applied them to develop a real-time implementation in the FPGA inside the software-defined radio we were using (USRP 2953r). All the relevant methods listed in chapters 6 and 7 were successfully implemented. Despite the unsolved problems

with FPGA - PC data transfer, we were able to confirm that our implementation of the techniques for the FHSS signal detection was working correctly. We proved that the use of the state-of-the-art SDR in real-time detection FHSS UAV's signals, possibly estimation of their parameters, is feasible. Providing a different point of view, we claim that our effort resulted in real-time spectrum analyzer with narrowband signal detection filter.

9.1 Future Work

Data Transfer: First, the author should fix the data transfer between the FPGA and host PC to work flawlessly and reliably.

Programming Skills: The author should also invest time to learn computer science skills to be able to create programs and FPGA designs swiftly and effectively.

Size of the DFT: FFT processor with different size should be implemented to add versatility. We propose the size of 256, 1024, or 2048 points.

Overlapping: Due to the use of windowing functions, time samples on the edge of the vector may be zero-forced, thus they have no effect on the DFT result. Computation of overlapping (Welch's method) DFT frames will mitigate this problem.

Polyphase Filtering: An alternative technique to FFT channelization might be polyphase filtering.

Improved CFAR processors: The proper analysis of the CFRA processors performance should be taken. Searching for more sophisticated CFAR methods like VIE-CFAR could bring an enhancement of the signal detection performance.

Time Difference of Arrival: Since AIT has several identical USRP 2953r devices at their disposal, we could reuse some of the well-known techniques solving a multilateration problems, GPS, etc. Those techniques may help to localize the position of the UAV controller transmitting the signal towards the drone.

Direction of Arrival: The localization problem may be partially solved by direction of arrival techniques like MUSIC, or ESPRIT, possibly their modification like Root MUSIC, or Unitary ESPRIT.

Fast WVD: We observed a decent performance of the PWVD time-frequency analysis technique. Its main problem was its high complexity, thus not implementable in the real-time scenario. Developing a fast algorithm solving the PWVD problem may refine the TFA of the received signal (compared to STFT).

Signal Classification: Synchronization to each hop of the FHSS signal may help in classification of such signal, potentially demodulation as even not decoded data may help in distinguishing between several FHSS transmitters operating in the sensed frequency band.

Reuse the Phase: In our project, we neglected the phase of channelized signal and focused on energy detection. We might exploit CORDIC algorithms to calculate the phases of the frequency bins and so obtain a new information about the sensed spectra.

Regressive Mathematics: Assuming our system provides a filtering FHSS narrowband signals from wide spectra, we significantly reduce the number of relevant samples. Over this reduced space of samples, we could apply machine learning algorithms to classify the signals, e.g., distinguishing between several transmitters.

Reactive Jamming: After refining our proposed real-time detection design, it could be used as first step for reactive jamming system.

References

[1] Parrot SA, *Parrot Mambo drone*, 2019. [Online]. Available: https://www.parrot.com/eu/drones/parrot-mambo-fpv{\#}parrot-mambo-fpv-details (visited on 03/01/2019) (page 33).

[2] ImmersionRC Ltd., *Immersion Vortex 250 Pro*, 2019. [Online]. Available: https://www.immersionrc.com/fpv-products/vortex-250-pro/ (visited on 03/01/2019) (page 33).

[3] SZ DJI Technology Co. Ltd., *DJI Mavic 2*, 2019. [Online]. Available: https://www.dji.com/at/mavic-2?site=brandsite{\&}from=nav (visited on 17/02/2019) (page 33).

[4] Intuitive Aerial AB, *No Title*, 2019. [Online]. Available: https://www.intuitiveaerial.com/aerigon/ (visited on 03/01/2019) (page 33).

[5] Flying-Cam, *No Title*. [Online]. Available: http://wp.flying-cam.com/ (visited on 03/01/2019) (page 33).

[6] DSLRPros Enterprise Drone Solutions, *DSLRPros Matrice 210*. [Online]. Available: https://www.dslrpros.com/dslrpros-matrice-210-xt2-first-responder-thermal-kit.html (visited on 03/01/2019) (page 33).

[7] AltiGator, *No Title*, 2019. [Online]. Available: https://altigator.com/police-support-drones/ (visited on 03/01/2019) (page 33).

[8] DSLRPros, *Agriculture & Farming Drones*. [Online]. Available: https://www.dslrpros.com/agriculture.html (visited on 03/01/2019) (page 33).

[9] Amazon Inc., *Amazon Prime Air*. [Online]. Available: https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8{\&}node=8037720011 (visited on 03/01/2019) (page 33).

[10] Z. Becvar, M. Vondra, P. Mach, J. Plachy and D. Gesbert, "Performance of Mobile Networks with UAVs: Can Flying Base Stations Substitute Ultra-Dense Small Cells?", in *European Wireless 2017; 23th European Wireless Conference*, 2017, pp. 1–7 (page 33).

[11] BBC, *Gatwick disruption: How will police catch the drone menace?*, 2018. [Online]. Available: https://www.bbc.com/news/technology-46648160 (visited on 13/01/2019) (page 34).

[12] OZYRPAS Consulting, *Global Drone Regulations Database*, 2019. [Online]. Available: https://www.droneregulations.info/ (visited on 03/01/2019) (page 34).

[13] Rohde & Schwarz GmbH, *Protecting the Sky Signal Monitoring of Radio Controlled Civilian Unmanned Aerial Vehicles and Possible Countermeasures*, 2019. [Online]. Available: http://www.rohde-schwarz-usa.com/rs/324-UVH-477/images/Drone{_}Monitoring{_}Whitepaper.pdf (visited on 03/01/2019) (page 34).

[14] PHANTOM Technologies Ltd., *EAGLE 108: DRONE DETECTION & JAMMING*, 2019. [Online]. Available: https://phantom-technologies.com/eagle108-drone-detection-jamming-system/ (visited on 03/01/2019) (page 34).

[15] Rohde & Schwarz GmbH, *R&S®ARDRONIS-I*, 2019. [Online]. Available: https://www.rohde-schwarz.com/uk/product/ardronis-i-productstartpage{_}63493-334548.html?change{_}c=true (visited on 03/01/2019) (page 34).

[16] Aaronia AG., *Aaronia AARTOS DDS*, 2019. [Online]. Available: https://www.aaronia.com/products/solutions/Aaronia-Drone-Detection-System/ (visited on 03/01/2019) (page 34).

[17] M. Jahangir and C. Baker, "Persistence surveillance of difficult to detect micro-drones with L-band 3-D holographic radar™", in *2016 CIE International Conference on Radar (RADAR)*, 2016, pp. 1–5. DOI: 10.1109/RADAR.2016.8059282 (page 34).

[18] RETIA Inc., *ReVISOR*, 2019. [Online]. Available: https://www.military-retia.eu/en/revisor (visited on 03/01/2019) (page 34).

[19] D. Torrieri, S. Talarico and M. C. Valenti, "Analysis of a Frequency-Hopping Millimeter-Wave Cellular Uplink", *IEEE Transactions on Wireless Communications*, 2016, ISSN: 15361276. DOI: 10.1109/TWC.2016.2597210. arXiv: arXiv:1607.08200v2 (pages 39, 42, 44).

[20] N. Telecommunications and I. Administration, *United States Frequency Allocation*, 2016. [Online]. Available: https://www.ntia.doc.gov/files/ntia/publications/january_2016_spectrum_wall_chart.pdf (visited on 17/02/2019) (page 41).

[21] Bluetooth Special Interest Group, *Bluetooth Core Specifications v5.0*, 2016. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification (pages 43, 45, 57, 79, 81, 87).

[22] D. Torrieri, *Principles of spread-spectrum communication systems*. 2015, ISBN: 9783319140964. DOI: 10.1007/978-3-319-14096-4 (page 44).

[23] Wikipedia, *ISM band*. [Online]. Available: en.wikipedia.org/wiki/ISM{_}band (visited on 17/02/2019) (page 44).

[24] ETSI, *EN 300 328 V2.1.1*, 2016. [Online]. Available: https://www.etsi.org/deliver/etsi_en/300300_300399/300328/02.01.01_60/en_300328v020101p.pdf (visited on 03/01/2019) (page 45).

[25] ——, *EN 302 502 V2.1.1*, 2017. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/edithelp/Docs/en_302502v211p_compared_with_previous_version.pdf (visited on 03/01/2019) (page 45).

[26] L. W. Matthew Loy Raju Karingattil, "ISM-Band and Short Range Device Regulatory Compliance Overview", 2005. [Online]. Available: http://www.ti.com/lit/an/swra048/swra048.pdf (visited on 17/02/2019) (page 45).

[27] U. L. Rohde and J. C. Whitaker, *Communications Receivers, Fourth Edition*. McGraw-Hill Education, 2017, ISBN: 9780071843348. [Online]. Available: https://books.google.cz/books?id=iVoYDgAAQBAJ (page 50).

[28] R. Svitek and S. Raman, "DC offsets in direct-conversion receivers: characterization and implications", *IEEE Microwave Magazine*, vol. 6, no. 3, pp. 76–86, 2005, ISSN: 1527-3342. DOI: 10.1109/MMW.2005.1511916 (page 50).

[29] M. Valkama, K. Salminen and M. Renfors, "Digital I/Q imbalance compensation in low-IF receivers: principles and practice", in *2002 14th International Conference on Digital Signal Processing Proceedings. DSP 2002 (Cat. No.02TH8628)*, vol. 2, 2002, 1179–1182 vol.2. DOI: 10.1109/ICDSP.2002.1028303 (page 50).

[30] G. Vallant, M. Epp, W. Schlecker, U. Schneider, L. Anttila and M. Valkama, "Analog IQ impairments in Zero-IF radar receivers: Analysis, measurements and digital compensation", in *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, 2012, pp. 1703–1707. DOI: 10.1109/I2MTC.2012.6229222 (page 50).

[31] Crunchbase, *Ettus Research acquired by National Instruments*, 2019. [Online]. Available: https://www.crunchbase.com/acquisition/national-instruments-acquires-ettus-research--4011011c (page 53).

[32] National Instruments, *CBX details*, 2018. [Online]. Available: https://kb.ettus.com/CBX (visited on 17/02/2019) (pages 54, 55).

[33] ——, "USRP-2953 Specifications", 2017. [Online]. Available: http://www.ni.com/pdf/manuals/374197d.pdf (visited on 17/02/2019) (pages 54, 55).

[34] Ettus Research, *USRP™ X300 and X310 X Series*. [Online]. Available: https://www.ettus.com/content/files/X300{_}X310{_}Spec{_}Sheet.pdf (visited on 17/02/2019) (pages 54, 55).

[35] ——, *RF front-end CBX scheme*, 2013. [Online]. Available: http://files.ettus.com/schematics/cbx/CBX.pdf (page 54).

[36] Xilinx Inc., *7 Series FPGAs Data Sheet: Overview*, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/data{_}sheets/ds180{_}7Series{_}Overview.pdf (visited on 17/02/2019) (page 54).

[37] ——, *DC and AC Switching Characteristics*, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/data{_}sheets/ds182{_}Kintex{_}7{_}Data{_}Sheet.pdf (visited on 17/02/2019) (page 54).

[38] National Instruments, *Switch Device for PCI Express*. [Online]. Available: http://www.ni.com/en-gb/support/model.cps-8910.html (visited on 17/02/2019) (page 54).

[39] ——, *SPXI Bus Extension Module*. [Online]. Available: http://www.ni.com/en-gb/support/model.pxie-8384.html (visited on 17/02/2019) (page 54).

[40] ——, *PXI Controller*. [Online]. Available: http://www.ni.com/en-gb/support/model.pxie-8880.html (visited on 17/02/2019) (page 54).

[41] J. Huang, Y. Zhou, N. Sun and Z. Hu, "Identification of Frequency-Hopping Spread Spectrum Signals Using SVMs with Second Generation Wavelet Kernels", in *2010 International Conference on Communications and Intelligence Information Security*, 2010, pp. 160–163. DOI: 10.1109/ICCIIS.2010.39 (page 57).

[42] L. Cohen, *Time-frequency Analysis*, ser. Electrical engineering signal processing. Prentice Hall PTR, 1995, ISBN: 9780135945322. [Online]. Available: https://books.google.cz/books?id=CSKLQgAACAAJ (pages 57, 65, 66).

[43] F. Hlawatsch and F. Auger, *Time Frequency Analysis: Concepts and Methods*. Wiley, 2010, ISBN: 9781848210332. DOI: 10.1002/9780470611203. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9780470611203 (pages 57, 66).

[44] H. Hassanieh, L. Shi, O. Abari, E. Hamed and D. Katabi, "GHz-wide sensing and decoding using the sparse Fourier transform", in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, 2014, pp. 2256–2264. DOI: 10.1109/INFOCOM.2014.6848169 (page 57).

[45] H. Hassanieh, "The Sparse Fourier Transform: Theory and Practice", PhD thesis, Massachusetts Institute of Technology, 2018, ISBN: 978-1-94748-707-9. DOI: 10.1145/3166186 (page 57).

[46] H. Hassanieh, P. Indyk, D. Katabi and E. Price, "Nearly Optimal Sparse Fourier Transform", *CoRR*, vol. abs/1201.2, 2012. arXiv: 1201.2501. [Online]. Available: http://arxiv.org/abs/1201.2501 (page 57).

[47] E. Sejdic, I. Orovic and S. Stankovic, "Compressive sensing meets time-frequency: An overview of recent advances in time-frequency processing of sparse signals", *Digital signal processing*, vol. 77, pp. 22–35, 2018 (page 57).

[48] A. Massa, "Compressive sensing - basics, state-of-the-art, and advances in Electromagnetic engineering", in *2015 9th European Conference on Antennas and Propagation (EuCAP)*, 2015, pp. 1–4 (page 57).

[49] X. Chen, L. Zhao and J. Li, "A modified spectrum sensing method for wideband cognitive radio based on compressive sensing", in *2009 Fourth International Conference on Communications and Networking in China*, 2009, pp. 1–5. DOI: 10.1109/CHINACOM.2009.5339762 (page 57).

[50] F. Liu, M. W. Marcellin, N. A. Goodman and A. Bilgin, "Compressive Sampling for Detection of Frequency-Hopping Spread Spectrum Signals", *IEEE Transactions on Signal Processing*, vol. 64, no. 21, pp. 5513–5524, 2016, ISSN: 1053-587X. DOI: 10.1109/TSP.2016.2597122 (page 57).

[51] Z. Vulaj and F. Kardovic, "Separation of sinusoidal and chirp components using Compressive sensing approach", *CoRR*, vol. abs/1502.0, 2015. arXiv: 1502.03628. [Online]. Available: http://arxiv.org/abs/1502.03628 (page 57).

[52] R. Hippenstiel, N. Khalil and M. Fargues, "The use of wavelets to identify frequency hopped signals", in *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems and Computers (Cat. No.97CB36136)*, vol. 1, 1997, 946–949 vol.1. DOI: 10.1109/ACSSC.1997.680583 (page 65).

[53] M Sirotiya and A Banerjee, "Detection and estimation of frequency hopping signals using wavelet transform", in *Cognitive Wireless Systems (UKIWCWS), 2010 Second UK-India-IDRC International Workshop on*, 2010, ISBN: 978-1-4244-9146-9. DOI: 10.1109/UKIWCWS.2010.5724233 (page 65).

[54] S. Barbarossa and A. Scaglione, "Parameter estimation of spread spectrum frequency-hopping signals using time-frequency distributions", *First IEEE Signal Processing Workshop on Signal Processing Advances in Wireless Communications*, 1997. DOI: 10.1109/SPAWC.1997.630288 (page 65).

[55] M. Gandetto, M. Guainazzo and C. S. Regazzoni, "Use of Time-Frequency Analysis and Neural Networks for Mode Identification in a Wireless Software-Defined Radio Approach", *EURASIP Journal on Advances in Signal Processing*, vol. 2004, no. 12, p. 863 653, 2004, ISSN: 1687-6180. DOI: 10.1155/S1110865704407057. [Online]. Available: https://doi.org/10.1155/S1110865704407057 (page 65).

[56] C. Chioncel, P. Chioncel, N. Gillich and O. Tirian, "Wigner Ville Distribution in Signal Processing, using Scilab Environment", *Analele Universitatii 'Eftimie Murgu' Resita*, vol. XVIII, pp. 101–106, 2011 (page 65).

[57] B. Barkat and B. Boashash, "A high-resolution quadratic time-frequency distribution for multicomponent signals analysis", *IEEE Transactions on Signal Processing*, vol. 49, no. 10, pp. 2232–2239, 2001, ISSN: 1053-587X. DOI: 10.1109/78.950779 (page 65).

[58] Y. Lei, Z. Zhong and Y. Wu, "A parameter estimation algorithm for high-speed frequency-hopping signals based on RSPWVD", in *2007 International Symposium on Intelligent Signal Processing and Communication Systems*, 2007, pp. 392–395. DOI: 10.1109/ISPACS.2007.4445906 (page 65).

[59] S. Shin Y. and J.-J. Jeon, *Pseudo Wigner-Ville Distribution, Computer program and its Applications to Time-Frequency Domain Problems*, 1993. [Online]. Available: https://calhoun.nps.edu/handle/10945/28793 (page 65).

[60] G. Andria, E. D'Ambrosio, M. Savino and A. Trotta, "Application of Wigner-Ville distribution to measurements on transient signals", in *1993 IEEE Instrumentation and Measurement Technology Conference*, 1993, pp. 612–617. DOI: 10.1109/IMTC.1993.382570 (page 65).

[61] G. Wei, T. Jiang, Y. Liu and W. Liu, "Dynamic gesture recognition using software defined radio based on smooth pseudo Wigner-Ville distribution", in *2015 9th International Conference on Signal Processing and Communication Systems (ICSPCS)*, 2015, pp. 1–5. DOI: 10.1109/ICSPCS.2015.7391721 (page 65).

[62] B. Boashash, "Time-Frequency Signal Analysis and Processing: A Comprehensive Reference", 2003 (page 66).

[63] F. Auger, E. Chassande-Mottin and P. Flandrin, "Making reassignment adjustable: The Levenberg-Marquardt approach", in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012, pp. 3889–3892. DOI: 10.1109/ICASSP.2012.6288767 (pages 66, 67).

[64] F. J. Harris, "On the use of windows for harmonic analysis with the discrete Fourier transform", *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978, ISSN: 0018-9219. DOI: 10.1109/PROC.1978.10837 (page 73).

[65] R. G. Lyons, *Understanding Digital Signal Processing, Third edition*. Pearson Education, 2010, ISBN: 9780137028528 (pages 74, 76, 77).

[66] A. Oppenheim and R. Schafer, *Discrete-time Signal Processing*, ser. Prentice-Hall signal processing series. Prentice Hall, 1989, ISBN: 9780132162920. [Online]. Available: https://books.google.cz/books?id=bPhSAAAAMAAJ (page 77).

[67] W. S. Smith, *The Scientist & Engineer's Guide to Digital Signal Processing*. California Technical Pub; 1st edition, 1997, ISBN: 0966017633 (page 77).

[68] E. Chu, E. Chu, R. Bond, C. Straub, A. George and O. Chemical Rubber Company (Cleveland, *Inside the FFT Black Box: Serial and Parallel Fast Fourier Transform Algorithms*, ser. Computational Mathematics Series. CRC-Press, 2000, ISBN: 9780849302701. [Online]. Available: https://books.google.cz/books?id=VjkZAQAAIAAJ (page 77).

[69] Cypress Semiconductor Corp., *Single-Chip Bluetooth Transceiver and Baseband Processor*, 2017. [Online]. Available: https://www.cypress.com/file/297961/download (visited on 17/02/2019) (page 82).

[70] Actions Technology, *ATS2815*. [Online]. Available: http://www.actions-semi.com/en/productview.aspx?id=231 (visited on 17/02/2019) (page 82).

[71] A. De Maio and M. S. Greco, *Modern Radar Detection Theory*, ser. Electromagnetics and Radar. Institution of Engineering and Technology, 2015, ISBN: 9781613531990. [Online]. Available: https://books.google.cz/books?id=ZKGiCwAAQBAJ (page 91).

[72] R. D. Hippenstiel, *Detection Theory: Applications and Digital Signal Processing*. CRC Press, 2017, ISBN: 9781351835947. [Online]. Available: https://books.google.cz/books?id=DApEDwAAQBAJ (page 91).

[73] H. Rohling, "Radar CFAR Thresholding in Clutter and Multiple Target Situations", *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-19, no. 4, pp. 608–621, 1983, ISSN: 0018-9251. DOI: 10.1109/TAES.1983.309350 (pages 91, 94).

[74] G. E. Vrckovnik and D. Faubert, "An Investigation of CFAR Techniques for Airborne Radars", p. 58, 1990 (pages 91, 94).

[75] C. Wolf, *Radar Basics: False Alarm Rate*. [Online]. Available: http://www.radartutorial.eu/index.en.html (visited on 17/02/2019) (page 91).

[76] Xilinx Inc., *DSP48 Macro v3.0*, 2015. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/xbip_dsp48_macro/v3_0/pg148-dsp48-macro.pdf (visited on 17/02/2019) (page 103).

[77] ——, *Fast Fourier Transform v9.0*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/xfft/v9_0/pg109-xfft.pdf (visited on 17/02/2019) (page 104).

[78] P. Kovář, *Družicová navigace: od teorie k aplikacím v softwarovém přijímači (Satellite Navigation: From Theory to Software-defined Radio Applications)*. Czech Technical University in Prague, 2016, ISBN: 978-80-01-05989-0.

[79] B. K. Levitt, U. Cheng, M. K. Simon and A. Polydoros, "Optimum Detection of Slow Frequency-Hopped Signals", *IEEE Transactions on Communications*, 1994, ISSN: 00906778. DOI: 10.1109/TCOMM.1994.583413.

[80] S. J. Orfanidis, *Applied Optimum Signal Processing*, Rutgers U. 2018, vol. 2. [Online]. Available: http://eceweb1.rutgers.edu/{~}orfanidi/aosp/.

[81] W. A. Gardner, "Exploitation of Spectral Redundancy in Cyclostationary Signals", *IEEE Signal Processing Magazine*, 1991, ISSN: 10535888. DOI: 10.1109/79.81007.

[82] K. Pärlin, M. M. Alam and Y. L. Moullec, "Jamming of UAV remote control systems using software defined radio", in *2018 International Conference on Military Communications and Information Systems (ICMCIS)*, 2018, pp. 1–6. DOI: 10.1109/ICMCIS.2018.8398711.

[83] F. J. Harris, C. Dick and M. Rice, "Digital receivers and transmitters using polyphase filter banks for wireless communications", *IEEE Transactions on Microwave Theory and Techniques*, vol. 51, no. 4, pp. 1395–1412, 2003, ISSN: 0018-9480. DOI: 10.1109/TMTT.2003.809176.

[84] Texas Instruments Inc., *CC2500 RF Transceiver*, 2019. [Online]. Available: http://www.ti.com/product/CC2500# (visited on 17/02/2019).

[85] Futaba Corporation, *Futaba FHSS Receivers*, 2019. [Online]. Available: https://www.futabarc.com/receivers/fhss.html (visited on 17/02/2019).

[86] Spektrum, Horizon Hobby, *Spektrum DSMX Hybrid Spread Spectrum*, 2019. [Online]. Available: https://www.spektrumrc.com/technology/DSMX.aspx (visited on 17/02/2019).

Appendix A

Appendix: Source Codes

All the scripts are fully working under the MATLAB v. 2017b. The scripts require measured and testing files, which will be provided on request.

A.1 FHSS Signal Simulation and TFA Comparisosn

```
%% FHSS simulation 2.0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   running on Lenovo X230
%   Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
%   RAM: 12288 MB
%   MATLAB 2017a
%
%   FEATURES
%   MFSK modulator including frequency hopping (FHSS) carrier modulator
%   output waveform generator
%   STFT spectrogram
%   Wavelet scalogram
%   Wigner-Wille distribution
%   Pseudo Wigner-Wille distribution
%   Smoothed Pseudo Wigner-Wille distribution
%   results saved into .eps figures
%
%   REQUIREMENTS:
%   Signal Processing Toolbox (mathworks)
%   Wavelet Toolbox (mathworks)
%   Time-Frequency Toolbox (Auger: http://tftb.nongnu.org/)
%
%   RECOMMENDATION:
%   run the script section by section
%
%   author: Skorepa (CTU:FEE, EURECOM, AIT student)
%   credits to: Mathworks toolboxes and Auger et al. for TFR toolbox

close all; clc; clear;

%% COLOURS

LINEWIDTH       = 1;
LINEWIDTH_AXIS  = 1;
```

```
BLUE_DEF        = [0 0.4470 0.7410];
ORANGE_DEF      = [0.8500 0.3250 0.0980];
YELLOW_DEF      = [0.9290 0.6940 0.1250];
PURPLE_DEF      = [0.4940 0.1840 0.5560];
GREEN_DEF       = [0.4660 0.6740 0.1880];
AQUA_DEF        = [0.3010 0.7450 0.9330];
RED_DEF         = [0.6350 0.0780 0.1840];
FONTSIZE        = 12;


%% Transmitter parameters

fbc     = 2450e6;    % FHSS band center frequency; carrier frequency (>= 2)
B_FHSS  = 100e6;     % FHSS bandwidth (not mislead it with FSK BW)
nhops   = 10;        % number of frequency hops within the band
kf      = 1/2;       % modulation index (depth), determines the frequency
                     % spacing
                     % min(kf) = 1/2 for MSK

res     = 2*B_FHSS;  % pick resolutoin w.r.t. fb (fc), remember Nyquist
                     % theorem
fs      = res;       % sampling frequency corresponding to the resolution
                     % of our time axis

T_hop  = 625e-6;       % in seconds,  625 us corresponds to the value of
                       % the Bluetooth standard
T_hopk = res * T_hop;  % hop duration in sampling space

bitrate = 1e6;        % bit rate of the data to-be-modulated signal



%% Time axis values

tot_nbits    = T_hop * nhops * bitrate; % number of transmitted bits during
                                        % the simulation
sim_duration =  nhops * T_hop;          % total duration of the simulation
                                        % in seconds;

t       = (0 : 1/(res) : sim_duration)';       % time axis

%% FHSS modulator

PNseq  = randi([ceil(fbc - B_FHSS/2) floor(fbc + B_FHSS/2)], nhops, 1);
seqhop = repelem(PNseq, T_hopk);
fc_hop = seqhop;

omega_c        = (2*pi*fc_hop.*t(1:end-1));  % carreir hopped angular freq.
hopped_carrier = cos(omega_c);               % carreir hopped signal

figure()
    plot(fc_hop,'s',...
    'LineWidth',2,...
    'MarkerSize',0.5,...
    'MarkerEdgeColor', BLUE_DEF,...
    'MarkerFaceColor', BLUE_DEF)
    grid on;
    xlim([0 length(fc_hop)])
    ylim([ceil(fbc - B_FHSS/2) floor(fbc + B_FHSS/2)])
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
```

```
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    view(90, 90)
    %title('hopping sequence in time')
    print('FHSS_sequence', '-depsc')
    print('FHSS_sequence', '-dpng')

figure()
    plot(real(hopped_carrier))
    grid on
    xlim([0 length(fc_hop)])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    xlabel('time [s]')
    ylabel('amplitude')
    %title('carrier freq')
    %print('FHSS_carrier', '-depsc')


%% M-FSK modulator
%   small recap:
%   M-FSK:    frequency shift keying - the most general term
%   M stands for the number of basis functions - several modulating
%   frequencies
%   CPFSK:  continuous phase FSK - avoiding abrupt changes of phase - its
%   discontinuity, which would corrupt the spectral properties - spectrum
%   broadening, PAPR
%   MSK:    special case of FSK with freqency separation equals to 2delta(f_c)
%   which equals to 0.5/T_s - this will preserve the orthogonality
%   (MSK operates with CP by default)
%   see Rappaport for equation references


%% generate digital binary modulating signal

%   generate input data bits
data_in  = randi([0 1], tot_nbits, 1);

%  creating binnary NRZ rec waveform: m(t) [Rapp 5.97]
 for i = 1 : 1 : tot_nbits

    %   create NRZ sequence within -1 to 1 range
    if (data_in(i) == 0)
        data_in(i) = -1;
    else
        data_in(i) = 1;
    end

 end

NRZ_rec = repelem(data_in, length(omega_c)/length(data_in));

%   theta(t) phase function of the modulating m(t) waveform
%   theta(t) = int_{-inf}^{t} m(tau) dtau
%   [Rapp 5.97]

NRZ_phase =  2 * pi * (cumtrapz(kf.*NRZ_rec)) ;

figure()
subplot(211)
    plot(NRZ_rec)
    grid on;
```

```
    xlim([0 length(NRZ_rec)])
    ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([-1 1])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('bit level', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    zoom on
subplot(212)
    plot(NRZ_phase)
    grid on;
    xlim([0 length(NRZ_rec)])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([ ])
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('phase in time', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('bits_phase', '-depsc')

% zoomed bits and corresponding phase
zoomed = 6000;
figure()
subplot(211)
    plot(NRZ_rec(1:zoomed))
    grid on;
    xlim([0 length(NRZ_rec(1:zoomed)) ])
    ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([-1 1])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('bit level', 'FontSize', FONTSIZE, 'FontWeight', 'bold')

subplot(212)
    plot(NRZ_phase(1:zoomed))
    grid on;
    xlim([0 length(NRZ_rec(1:zoomed)) ])
    %ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([ ])
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('phase in time', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('bits_phase_detail', '-depsc')



%% modulating the carrier with binary NRZ signal

s_n    =  exp(1i*(omega_c + NRZ_phase)); % FSK

figure()
plot(real(s_n))
    grid on
    xlim([0 length(fc_hop)])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
```

```
    xlabel('time [s]')
    ylabel('amplitude')
    print('bit_mod_signal', '-depsc')

% zoomed modulated and hopped signal
zoomstart = 20150;
zoomend   = 20250;

figure()
% subplot(311)
%     plot(real(hopped_carrier(zoomstart:zoomend)) )
%     grid on
%     xlim([0 length(fc_hop(zoomstart:zoomend)) ])
%     xt = get(gca, 'XTick');
%     set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
%     ylim([-1.5 1.5])
%     xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
%     ylabel('amplitude', 'FontSize', FONTSIZE, 'FontWeight', 'bold')


subplot(211)
    plot(NRZ_rec(zoomstart:zoomend))
    grid on;
    xlim([0 length(NRZ_rec(zoomstart:zoomend)) ])
    ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([-1 1])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('bit level', 'FontSize', FONTSIZE, 'FontWeight', 'bold')

subplot(212)
    plot(real( s_n(zoomstart:zoomend)))
    grid on
    xlim([0 length( s_n(zoomstart:zoomend)) ])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    ylim([-1.5 1.5])
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('amplitude', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('bit_mod_signal_detail', '-depsc')



%% adding noise

SNR = 5;
s_n  = awgn(s_n, SNR, 'measured', 'dB');


%% Set the window FFT parameters and compute the spectrum

nbins      = 512;       % size of the taken DFT
select_win = 10;        % choose windowing funciton

winlist = {@barthannwin, @bartlett, @blackman, @blackmanharris, ...
    @bohmanwin, @chebwin, @flattopwin, @gausswin, @hamming, @hann, ...
    @kaiser, @nuttallwin, @parzenwin, @rectwin, @tukeywin, @triang};

win = window(winlist{select_win}, nbins);
```

```
FFT.s_n          = zeros(floor( length(s_n)/nbins), nbins);
FFT.s_n_onesided = zeros(floor( length(s_n)/nbins), nbins/2);

tic
for p = 1 : 1 : floor(length(s_n)/nbins)

    windowed_st   = win.*s_n( (p-1)*nbins + 1 : (p-1)*nbins + nbins );
    FFT.s_n(p, :) = ( fft( real(windowed_st), nbins) / nbins);

    FFT.s_n_onesided(p, :) = (FFT.s_n(p, 1:nbins/2));

end
toc

 %% Plot FFT results

figure()
imagesc( 20*log10( fliplr( abs(FFT.s_n_onesided) ) ) )  % orientation t0 = top,
                                                        % t_last = bottom
    colorbar
    colormap('jet')
    xticks([0 nbins/8 nbins/4 (nbins/4+nbins/8) nbins/2]) % set xaxis ticks
    xt  = get(gca, 'XTick');                              % relabel xticks
    xt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'XTick', xt, 'XTickLabel', xt2);
    xtickangle(30)                                        % rotate xticks
    yt  = get(gca, 'YTick');                              % set yaxis ticks
    yt2 = (yt*nbins)/(T_hopk/T_hop);                      % relabel yticks
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ylabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    xlabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    hcb = colorbar;
    title(hcb, 'log-magnitude')
    print('512_FFT_Hann', '-depsc')


% plot detailed results
zoomstart = 1;
zoomend   = 5000;
figure()
    plot(NRZ_rec(zoomstart:zoomend))
    grid on;
    xlim([0 length(NRZ_rec(zoomstart:zoomend)) ])
    ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([-1 1])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('bit level', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('NRZ_rec_detail_3', '-depsc')

figure()
    imagesc( 20*log10( fliplr( abs(FFT.s_n_onesided(…
        zoomstart:ceil(zoomend/nbins), :)) ) ) )  % orientation t0 = top,
                                                  % t_last = bottom
    colormap('jet')
    xticks([0 nbins/8 nbins/4 (nbins/4+nbins/8) nbins/2]) % set xaxis ticks
    xt  = get(gca, 'XTick');                              % relabel xticks
    xt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
```

```
    set(gca, 'XTick', xt, 'XTickLabel', xt2);
    xtickangle(30)                                          % rotate xticks
    yt  = get(gca, 'YTick');                                % set yaxis ticks
    yt  = yt(1:2:end);
    yt2 = (yt*nbins)/(T_hopk/T_hop);                        % relabel yticks
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ylabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    xlabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    view(-90, -90)
    print('512_STFT_detail_2', '-dpng')
    print('512_STFT_detail_2', '-depsc')


%% emptying RAM

clear FFT
close all;


%% WAVELET TRANSFORM

F      = 32;
a0     = 2^(1/F);
scales = 2*a0.^(0:5*F);

[cfs,frequencies] = cwt(s_n, scales, 'cmor1-1.5', (1/fs));


%% PLOT WAVELET TRANSFORM

figure()
    imagesc( t(1:end-1), 0 : 1 : B_FHSS, abs(cfs));
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    colorbar
    colormap('jet')
    hcb = colorbar;
    title(hcb, 'lin-magnitude')
    view(-90, -90)
    print('wavelet', '-depsc')


%% PLOT WAVELET OF USE

N = 5000;
Lb = -4;
Ub = 4;
fb = 1;
fc = 1.5;
[psi,x] = cmorwavf(Lb,Ub,N,fb,fc);

figure()
%subplot(2,1,1)
    plot(x,real(psi), ...
                'LineWidth', 1.5)
    grid on
    xt  = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', []);
    xlabel('complex morlet wavelet', 'FontSize', FONTSIZE, 'FontWeight', 'bold');
    hold on
%subplot(2,1,2)
    plot(x,imag(psi), ...
```

```
        'LineWidth', 1.5)
    legend('Re(cmor)', 'Im(cmor)')
%     grid on;
%     xt  = get(gca, 'XTick');
%     set(gca, 'XTick', xt, 'XTickLabel', []);
%     xlabel('Imaginary Part', 'FontSize', FONTSIZE, 'FontWeight', 'bold');
     print('used_wavelet', '-depsc')

%% WIGNER VILLE (AUGER)

comp_chunk = 1e5;                   % computing the PWVD of the whole signal
                                    % would consume your RAM, therefore we
                                    % need to split into smaller chunks
X          = s_n;
T          = 1:comp_chunk;
N          = 512;
TRACE      = [];

pure_wvd = zeros(N, length(X));

for k = 1 : 1 : floor(length(X)/comp_chunk)

fprintf('WVD computation: chunk %d of %d\n', k,  floor(length(X)/comp_chunk))

pure_wvd(:, (k-1)*comp_chunk + 1 : k*comp_chunk )  = ...
    ( tfrwv( X((k-1)*comp_chunk + 1 : k*comp_chunk), T, N, TRACE));

end

T = 1:rem(length(X), comp_chunk);

fprintf('WVD computation of the remainder\n')

pure_wvd(:, k*comp_chunk + 1 : k*comp_chunk + rem(length(X), comp_chunk) ) = ...
     ( tfrwv( X(k*comp_chunk + 1 : ...
    k*comp_chunk + rem(length(X), comp_chunk)), T, N, TRACE));

disp('WVD computation done')

%% Plot WVD

figure()
imagesc(abs(pure_wvd))
    yticks([0 N/4 N/2 (N/2+N/4) N]) % set xaxis ticks
    yt  = get(gca, 'YTick');                                % relabel xticks
    yt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ytickangle(30)
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    colorbar
    colormap('jet')
    hcb = colorbar;
    title(hcb, 'lin-magnitude')
    view(-90, -90)
    print('512_WVD_low_SNR_2', '-depsc')

% plot detailed results
zoomstart = 1;
zoomend   = 5000;
```

```
figure()
    plot(NRZ_rec(zoomstart:zoomend))
    grid on;
    xlim([0 length(NRZ_rec(zoomstart:zoomend)) ])
    ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([-1 1])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('bit level', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('NRZ_rec_detail_2', '-depsc')

 figure()
    imagesc( real( (pure_wvd(:,zoomstart:zoomend))))
    yticks([0 N/4 N/2 (N/2+N/4) N]) % set xaxis ticks
    yt  = get(gca, 'YTick');                                % relabel xticks
    yt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ytickangle(30)
    xt = get(gca, 'XTick');
    xt  = xt(2:2:end);
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    %colormap('jet')
    print('512_WVD_low_SNR_detail_2', '-depsc')


%% PLOT TRUE WVD with cross-term demonstration
%
% in this section, we use WVD only for the hopped carrier (without
% FSK modulated data) as the original simulation would be too large for
% computation of the WVD

seqhop = repelem(PNseq, T_hopk/100);  %
fc_hop = seqhop;

omega_c        = (2*pi*fc_hop.*t(1:1:length(fc_hop)));
hopped_carrier = cos(omega_c);

hopped_carrier2 = awgn(hopped_carrier, 5, 'measured', 'dB');

wvd_s_hop = tfrwv(hilbert(hopped_carrier));

A = length(hopped_carrier)

figure()
imagesc( abs( wvd_s_hop))
    yticks([0 A/4 A/2 (A/2+A/4) A]) % set xaxis ticks
    yt  = get(gca, 'YTick');                                % relabel xticks
    yt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ytickangle(30)
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', 100*xt/(T_hopk/T_hop))
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    colorbar
    colormap('jet')
    hcb = colorbar;
```

```
    title(hcb, 'lin-magnitude')
    view(-90, -90)
    print('WVD_s_hopped', '-depsc')


%% emptying RAM

clear pure_wvd
close all


%% PSEUDO WIGNER VILLE (AUGER)

comp_chunk = 1e5;                   % computing the PWVD of the whole signal
                                    % would consume your RAM, therefore we
                                    % need to split into smaller chunks
X          = s_n;
T          = 1:comp_chunk;
N          = 512;
H          = window(@rectwin,N/3);
TRACE      = [];

pwvd = zeros(N, length(X));

for k = 1 : 1 : floor(length(X)/comp_chunk)

fprintf('PWVD computation: chunk %d of %d\n', k,  floor(length(X)/comp_chunk))

pwvd(:, (k-1)*comp_chunk + 1 : k*comp_chunk )  = ...
    ( tfrpwv( X((k-1)*comp_chunk + 1 : k*comp_chunk), T, N, H, TRACE));

end

T = 1:rem(length(X), comp_chunk);

fprintf('PWVD computation of the remainder\n')

pwvd(:, k*comp_chunk + 1 : k*comp_chunk + rem(length(X), comp_chunk) ) = ...
     ( tfrpwv( X(k*comp_chunk + 1 : ...
    k*comp_chunk + rem(length(X), comp_chunk)), T, N, H, TRACE));

disp('PWVD computation done')


%% Plot PWVD

figure()
imagesc(abs(pwvd))
    yticks([0 N/4 N/2 (N/2+N/4) N]) % set xaxis ticks
    yt  = get(gca, 'YTick');                                % relabel xticks
    yt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ytickangle(30)
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    colorbar
    colormap('jet')
    hcb = colorbar;
    title(hcb, 'lin-magnitude')
    view(-90, -90)
```

```
    print('512_PWVD_low_SNR_2', '-depsc')

% plot detailed reuslts
zoomstart = 1;
zoomend   = 5000;
figure()
    plot(NRZ_rec(zoomstart:zoomend))
    grid on;
    xlim([0 length(NRZ_rec(zoomstart:zoomend)) ])
    ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([-1 1])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('bit level', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('NRZ_rec_detail_2', '-depsc')


figure()
    imagesc( real( (pwvd(:,zoomstart:zoomend))))
    yticks([0 N/4 N/2 (N/2+N/4) N]) % set xaxis ticks
    yt  = get(gca, 'YTick');                              % relabel xticks
    yt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ytickangle(30)
    xt = get(gca, 'XTick');
    xt  = xt(2:2:end);
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    %colormap('jet')
    print('512_PWVD_low_SNR_detail_2', '-depsc')


%% emptying RAM

clear pwvd
close all

%% SMOOTHED PSEUDO WIGNER VILLE (AUGER)

comp_chunk = 1e4;                   % computing the PWVD of the whole signal
                                    % would consume your RAM, therefore we
                                    % need to split into smaller chunks
X          = s_n;
T          = 1:comp_chunk;
N          = 512;
H          = window(@rectwin,N/3);
G          = window(@hamming,N/3);
TRACE      = [];

spwvd = zeros(N, length(X));

tic
for k = 1 : 1 : floor(length(X)/comp_chunk)

fprintf('SPWVD computation: chunk %d of %d\n', k,  floor(length(X)/comp_chunk))

spwvd(:, (k-1)*comp_chunk + 1 : k*comp_chunk )  = ...
    ( tfrspwv( X((k-1)*comp_chunk + 1 : k*comp_chunk), T, N, H, G, TRACE));
```

```
end

T = 1:rem(length(X), comp_chunk);

disp(sprintf('SPWVD computation of the remainder'))

spwvd(:, k*comp_chunk + 1 : k*comp_chunk + rem(length(X), comp_chunk) ) = ...
     ( tfrspwv( X(k*comp_chunk + 1 : ...
    k*comp_chunk + rem(length(X), comp_chunk)), T, N, H, G, TRACE));

disp('SPWVD computation done')
toc

%% Plot SPWVD

figure()
imagesc(abs(spwvd))
    yticks([0 N/4 N/2 (N/2+N/4) N]) % set xaxis ticks
    yt  = get(gca, 'YTick');                                % relabel xticks
    yt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ytickangle(30)
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    colorbar
    colormap('jet')
    hcb = colorbar;
    title(hcb, 'lin-magnitude')
    view(-90, -90)
    print('512_SPWVD_low_SNR_2', '-depsc')

% plot detailed reuslts
zoomstart = 1;
zoomend   = 5000;
figure()
    plot(NRZ_rec(zoomstart:zoomend))
    grid on;
    xlim([0 length(NRZ_rec(zoomstart:zoomend)) ])
    ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([-1 1])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('bit level', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('NRZ_rec_detail_2', '-depsc')


figure()
    plot(NRZ_phase(zoomstart:zoomend))
    grid on;
    xlim([0 length(NRZ_phase(zoomstart:zoomend)) ])
    %ylim([-1.5 1.5])
    xt = get(gca, 'XTick');
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    yticks([ ])
    yt = get(gca, 'YTick');
    set(gca, 'YTick', yt, 'YTickLabel', (yt+1)/2  )
```

```
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('phase', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    print('NRZ_phase_detail_2', '-depsc')

figure()
    imagesc( real( (spwvd(:,zoomstart:zoomend))))
    yticks([0 N/4 N/2 (N/2+N/4) N]) % set xaxis ticks
    yt  = get(gca, 'YTick');                                % relabel xticks
    yt2 = [0  B_FHSS/4 B_FHSS/2 B_FHSS/2+B_FHSS/4 B_FHSS];
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ytickangle(30)
    xt = get(gca, 'XTick');
    xt  = xt(2:2:end);
    set(gca, 'XTick', xt, 'XTickLabel', xt/(T_hopk/T_hop))
    xlabel('time [s]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    ylabel('frequency [Hz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    %colormap('jet')
    print('512_SPWVD_low_SNR_detail_2', '-depsc')


%% emptying RAM

clear spwvd
close all
```

A.2 Loading Recorded Data Function

```
function [rec_data, rec_time] = LVbin2matlabdata(fname, nsamples, fs)
%% LabVIEW binary file to matlab data
%   This function is used to convert data gaiend from LabVIEW
%   Communications 2.0 Tool for further processing in MATLAB.
%   The gained data are continuously measured waveform samples fetched from
%   a USRP.
%
%   The function s inputs are the filename "fname" of the .bin/.dat file
%   comming from LabVIEW and number of samples "nsamples", wchich were
%   fetcehd from the FGPA s FIFO memory at once - in one chunk.
%   To determine the total time of the measurement record, the
%   sampling rate "fs" is also needed.
%
%   The function returns recorded data "rec_data" and total time of the
%   measurement record "rec_time" in seconds.
%   Also, the 'wave_data' is the unprocessed matrix measured by the
%   LabVIEW. It has its original format U32 even the matlab returns this as
%   a double. Hence you might find convenient to convert it back to U32
%   format by simple 'uint32(wave_data)' implicit matlab function.
%
%   The bin/dat LabVIEW comm output file has following size:
%   number of rows :       invokes the number of samples taken at once from
%                          the FPGA s FIFO memory
%                          IMPORTANTT: the first row of the file refers
%                          only to the size of the chunk which is fetched
%                          from FPGA i.e. the "number_of_samples"
```

```
%   number of columns :    refers to the total time of the taken measurement,
%                          in other words it is equal to the total number of
%                          collected samples divided by the number of samples
%                          taken at once
%   ------------------------------------------------------------------------
%   AIT, David L schenbrand, Jiri Skorepa
%   History:  20180927 DLoe: create file
%   ------------------------------------------------------------------------

fid           =  fopen(fname);
finfo         =  dir(fname);
disp(sprintf('OPENING DATA FILE WITH THE SIZE OF %d KB. PLEASE WAIT…', …
                        (finfo.bytes/1024)))
wave_data     =  fread(fid, [nsamples + 1, (finfo.bytes/(nsamples+1)/4) ], …
                        'uint32', 'b');
% ETTUS USRPs: the output waveform data are in unsigned integer format,
% 32-bit chunks, big-endian
% Windows uses little-endian, I think
% inside the fread function, the size of the wave_data matrix is being
% defined - the number of columns is computed from the nsamples known input
% and the byte size of the whole file
% after all, the LabVIEW file is uint32, which is 4 bytes
disp('DATA OPENED AND READ.')
fclose(fid);

rec_time =  (size(wave_data, 1)*size(wave_data,2)) / fs;


num_I = hex2dec('0000FFFF'); % masking I data
num_Q = hex2dec('FFFF0000'); % masking Q data

% allocate the exact of memory for storing the splitted I and Q components and
% resulting data (double data type costs more memory)
I_data    = zeros(nsamples,size(wave_data, 2), 'int16');
Q_data    = zeros(nsamples,size(wave_data, 2), 'int16');
rec_data  = zeros(nsamples,size(wave_data, 2), 'int16');


cnt       = 0.1;
title_str = 'LOADING DATA INTO MATLAB WORKSPACE: %d%% DONE';
disp('LOADING DATA INTO MATLAB WORKSPACE...')

for i = 1 : 1 : size(wave_data, 2)

    % the first row of I_data, Q_data, tmp invokes the column size of the
    % vector = number of samples taken from the FPGA s FIFO memory at once
    I_data(:,i) = typecast(uint16(bitand(wave_data(2:nsamples+1,i), …
                                num_I)),'int16');
    % The 4 rightmost (lower, LSB) bits correspond to I
    Q_data(:,i) = typecast(uint16(bitand(wave_data(2:nsamples+1,i), …
                                num_Q)/2^16),'int16');
    % The 4 leftmost (higher, MSB) bits correspond to Q
    rec_data(:,i)    = complex(I_data(:,i),Q_data(:,i));

    if i == ceil(size(wave_data, 2)*cnt)
        disp(sprintf(title_str, uint16(cnt*100)))
        cnt = cnt + 0.1;
    end

end
```

A.3 Analysis of The Recorded Signal

```
%% FHSS measurement analysis
%   MATLAB R2017a
%   running on Lenovo X230
%   Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
%   RAM: 12288 MB
%
%   Reading from the LabVIEW binary file containgn USRP record of FPGA
%   FFTed data
%
%   REQUIREMENTS:
%   function: LVbin2matlabdata.m
%
%   WARNING: Make sure you have enough RAM
%
%   RECOMMENDATION: run the script section by section

close all; clc; clear;

%% COLOURS

LINEWIDTH       = 1;
LINEWIDTH_AXIS  = 1;
BLUE_DEF        = [0 0.4470 0.7410];
ORANGE_DEF      = [0.8500 0.3250 0.0980];
YELLOW_DEF      = [0.9290 0.6940 0.1250];
PURPLE_DEF      = [0.4940 0.1840 0.5560];
GREEN_DEF       = [0.4660 0.6740 0.1880];
AQUA_DEF        = [0.3010 0.7450 0.9330];
RED_DEF         = [0.6350 0.0780 0.1840];
FONTSIZE        = 12;

%% loading data from the binary file

sample_rate = 100e6;    % sampling frequency set in LabVIEW
nsamples    = 100024;   % number of samples, which were transfered between
                        % the FPGAs FIFO memory and comp memory at one chunk

filename    = 'meas_20_100MHz_DJI6_fixed_fetching.bin';
% use files 'meas_21_100MHz_BT_161018.bin'
% or 'meas_20_100MHz_DJI6_fixed_fetching.bin'

[recorded_data, recording_time] = LVbin2matlabdata(filename, nsamples, sample_rate);


%% reshaping received data array into one vector of consecutive samples
% also convert in double for futuer Matlab computations
%recorded_data = wave_data;

rec_samples = double(reshape ...
    (recorded_data, 1, size(recorded_data, 1) * size(recorded_data, 2)));

disp(sprintf('YOUR RECORDED SIGNAL IS %d SECONDS LONG', recording_time));


%% emptying RAM

clear recorded_data

%% computing the FFT and plot the results
```

```
BW            = 100e6;          % signal bandwidth
fs            = 2*BW;            % fs  = sample_rate

winlist = {@barthannwin, ...     % 1
           @bartlett, ...        % 2
           @blackman, ...        % 3
           @blackmanharris, ...  % 4
           @bohmanwin, ...       % 5
           @chebwin, ...         % 6
           @flattopwin, ...      % 7
           @gausswin, ...        % 8
           @hamming, ...         % 9
           @hann, ...            % 10
           @kaiser, ...          % 11
           @nuttallwin, ...      % 12
           @parzenwin, ...       % 13
           @rectwin, ...         % 14
           @tukeywin, ...        % 15
           @triang};             % 16
wf      = 10;                    % select the window from the list
nbins   = 512;                   % size of the taken DFT
nrows   = 10e3;
chunk   = nbins * nrows;
df = 200e6/nbins;
dt = 1/200e6;

fprintf('PLOTTING THE SIGNAL TIME INTERVAL LENGTH: %d seconds\n', chunk*1/BW)
fprintf('TIME RESOLUTION: %d seconds\n', chunk*1/BW)
fprintf('FREQUENCY RESOLUTION: %d Hz\n', BW/nbins )

win = window(winlist{wf}, nbins)';

s_n     = zeros(1, chunk);
FFT.s_n = zeros(nrows, nbins);

figure()

for k = 1 : 1 : floor(length(rec_samples)/chunk)

    fprintf('TIME INTERVAL CHUNK %d of %d\n',k,floor(length(rec_samples)/chunk))
    s_n = rec_samples( (k-1)*chunk + 1 : k*chunk);

    for kk = 1 : 1 : nrows

        windowed_st   = win.*s_n((kk-1)*nbins + 1 : kk*nbins );
        FFT.s_n(kk, :) = 10*log10( abs( fftshift(fft( (windowed_st), nbins))*dt));

    end

    imagesc(FFT.s_n)
    colorbar
    colormap('jet')
    xticks([0 nbins/4 nbins/2 (nbins/2+nbins/4) nbins]) % set xaxis ticks
    xt  = get(gca, 'XTick');                            % relabel xticks
    xt2 = [0  BW/4 BW/2 BW/2+BW/4 BW]/1e6;
    set(gca, 'XTick', xt, 'XTickLabel', xt2);
    %xtickangle(30)                                     % rotate xticks
    yt  = get(gca, 'YTick');                            % set yaxis ticks
    yt2 = (yt*nbins/1e2);                               % relabel yticks
    set(gca, 'YTick', yt, 'YTickLabel', yt2);
    ylabel('time [us]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
```

```
    xlabel('frequency [MHz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    hcb = colorbar;
    title(hcb, 'log-magnitude')

    %print('meas_BT_spec_512_DFT_hann', '-depsc')
    disp('PRESS A KEY TO CONTINUE TO THE NEXT CHUNK')
    pause;

end

%% plotting time domain signal detail
% if you analyze the file 'meas_21_100MHz_BT_161018.bin', you may try to
% zoom in the clearly visible hops (thanks to low FSL and thus high SNR)
% zoom in the begining of the hop and see the two diferent modulation
% formats - packet header = GFSK, payload = 8PSK
%
% in case of analysis of the 'meas_20_100MHz...bin', you may see the GFSK
% modulation of the FHSS signal hops

chunk2 = chunk;
fprintf('PLOTTING THE SIGNAL TIME INTERVAL LENGTH: %d seconds\n', chunk*1/BW)

figure()
for k = 1 : 1 : floor(length(rec_samples)/chunk2)

    fprintf('TIME INTERVAL CHUNK %d of %d\n',k,floor(length(rec_samples)/chunk))
    s_n = rec_samples( (k-1)*chunk2 + 1 : k*chunk2);

    plot((1:chunk2)/100e6, real(s_n))
        xlim([0 chunk2/100e6])
        xlabel('time [seconds]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
        ylabel('amplitude', 'FontSize', FONTSIZE, 'FontWeight', 'bold')

    %print('meas_DJI_TD', '-depsc')
    disp('PRESS A KEY TO CONTINUE TO THE NEXT CHUNK')
    pause;
end


%% plotting frequency domain signal detail - ONE DFT frame


df = 200e6/nbins;
dt = 1/200e6;

start1 = 1;

fprintf('PLOTTING THE SIGNAL TIME INTERVAL LENGTH: %d seconds\n', nbins*1/BW);

figure()
for p = start1 : 1 : floor(length(rec_samples)/nbins)

    fprintf('TIME INTERVAL CHUNK %d of %d\n',p,floor(length(rec_samples)/nbins))
    s_n = rec_samples( (p-1)*nbins + 1 : p*nbins);

    windowed_st = win.*s_n;
    FFT.detail  = 10*log10( abs( fftshift( fft( windowed_st, nbins)*dt)));
    %fshift      = (-nbins/2 : nbins:2 -1) * (fs/nbins);

    plot(1:512, FFT.detail)
        grid on
        xlim([0 512])
```

```
        xticks([0 nbins/4 nbins/2 (nbins/2+nbins/4) nbins]) % set xaxis ticks
        xt  = get(gca, 'XTick');                            % relabel xticks
        xt2 = [0  BW/4 BW/2 BW/2+BW/4 BW]/1e6;
        set(gca, 'XTick', xt, 'XTickLabel', xt2);
        ylabel('log-magnitude', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
        xlabel('frequency [MHz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')

    %print('meas_DJI_one_512_DFT_hann', '-depsc')
    disp('PRESS A KEY TO CONTINUE TO THE NEXT CHUNK')
    pause;

end
```

A.4 CFAR Threshold Processors Funciton

```
%% ADAPTIVE THRESHOLD BASED ON CFAR ALGORITHMS
%   This algorithm computes the threshold for the further detection for
%   your variant data. In an intuitive way, the algorithm works like that:
%   You want to compute a threshold for a value at 'n' position. Hence, you
%   take a look on values at your right side and left side neigbouring positions.
%   From those values, you compute basic statistic. You can choce between
%   mean (CA), median (OS) or you take a neighbour of your 'n' position
%   under test  with Kth largest value  as your threshold.
%   You repeat this operation for each 'n' position value of your data
%   matrix
%
%   Constant False Alarm Ratio
%   CA-CFAR: Cell Average CFAR = you are taking mean of your neighbouring
%   cells
%   CAGO-CFAR: Cell Average Greatest Of = both neighbouring windows/cells
%   are averaged separately, then the higher result is taken as a result
%   CASO-CFAR: Cell Average Smallest OF = both neighbouring windows/cells
%   are averaged separately, then the smaller result is taken as a result
%   OS-CFAR: Order Statistics CFAR = sort your neigbours according to
%   their value - from min to max, then take Kth max neighbour as your
%   trheshold
%   OR
%   you can take median of your sorted negibours
%
%   threshold is returend in logarithmic scale abs 20 log10
%   INPUTS:
%   data:   your data matrix for which you want to compute the threshold
%   win:    number neighbouring cells/positions which should be considered
%           for the thershold computation. Only one side, so multiplied by
%           2 in total for both sides - left and right
%   gr:     number of guarding cells, the very adjacent neighbours to your
%           position/cell under test, which will NOT be considered for the
%           computation
%   K:      Kth largest neigbour from the sorted neighbour set, for OS-CFAR
%   scale:  scaling of your threshold to fit
%   select: CFAR function select. You can chose between the median of your
%           neighbouring cells - select = 1, mean - select = 2, or Kth
%           largest one - select = 3
%
%
%   OUTPUTS:
%   thr:    is computed threshold corresponding to your input 'data' matrix
%           ////it is returend in logatirhmic scale abs(20log10(data))
```

```
function thr = cfarthres2(data, win, gr, K, scale, select)

data      = abs(data);
ww        = win;
gr        = gr;
K         = K; % Kth largest sample from neighbouring samples
scale     = scale;
stat_func = {@mean, @maxk, @median};

win_neigh  = zeros(1, 2*ww);
thr        = zeros(size(data, 1), size(data, 2));

cfar_funcs = {'CA-CFAR', 'OS-CFAR', 'CAGO-CFAR', 'CASO-CFAR'};

disp('.....................................')
disp('DATA LOADED')
disp(sprintf('FUNCTION: %s', cfar_funcs{select}))
disp(sprintf('NUMBER OF NEIGBOURING CELLS: %d', ww))
disp(sprintf('NUMBER OF GUARDING CELLS: %d', gr))
disp(sprintf('Kth FACTOR: %d', K))
disp(sprintf('SCALE: %.5f', scale))
disp('.....................................')
disp('COMPUTING THRESHOLD FOR YOUR DATA MATRIX...')

cnt       = 0.1;
title_str = 'COMPUTING THRESHOLD %d%% DONE';

for n = 1 : 1 : size(data, 1)

    row_data     = data(n, :);
    ext_row_data = cat(2, row_data(length(row_data) - …
         (ww + gr - 1) : end), row_data, row_data(1 : ww + gr));

    for k = 1 : 1 : size(row_data, 2)

            win_neigh_L = ext_row_data(k : k + ww-1);
                        % left side neighbouring window/cell
            win_neigh_R = ext_row_data(ww + 2*gr + k + 1 : 2*ww + 2*gr + k);
                        % right side neighbouring window/cell
            win_neigh   = cat(2, win_neigh_L, win_neigh_R);
                        % concatenation of both windows/cells


            if select == 2
                sorted_win = scale * (sort(win_neigh));
                thr(n, k)  = sorted_win(K);
            elseif select == 4
                thr(n, k) = scale * max(mean(win_neigh_L), mean(win_neigh_R));
            elseif select == 5
                thr(n, k) = scale * min(mean(win_neigh_L), mean(win_neigh_R));
            else
                thr(n, k) = scale * stat_func{select}(win_neigh);
            end

    end

    if n == ceil(size(data, 1)*cnt)
       disp(sprintf(title_str, uint16(cnt*100)))
       cnt = cnt + 0.1;
    end
```

```
end
```

A.5 CFAR Comparison Script

```
%% CFAR DEMONSTRATION
%   MATLAB R2017a
%   requirements:
%   matrices containing the data:
%       meas_20_WIFI_long.mat
%       meas_20_WIFI_contained.mat
%       meas_20_DJI_WIFI_long.mat
%       meas_20_DJI_pulse_WIFI_contained.mat
%       meas_20_DJI_longlong.mat
%       meas_20_DJI_long.mat
%   cfarthres.m function
%
%   running on Lenovo X230
%   Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
%   RAM: 12288 MB


clc; close all; clear;


%% load data

DJI_pulse3      = load('meas_20_DJI_pulse_contained.mat');

DJI_pulse2      = load('meas_20_DJI_pulse_WIFI_contained.mat');

% very simple way how to create an signal including WIFI and drone's
% narrow pulse, for the demonstration, it is OK
DJI_pulse.pokus =  DJI_pulse2.pokus+DJI_pulse3.pokus/7;


%% load and add noise

%SNR = -26;
SNR = 60;

noise.DJI            = awgn(DJI_pulse.pokus, SNR, 'measured', 'dB');


%% switch between raw and noise-added data

do_you_want_noise_added = 'yes'

if do_you_want_noise_added == 'yes'

    DJI_pulse.pokus      = noise.DJI;

end

 %% FFT of the data

nbins = 512;
w  = window(@hann,nbins)';

df = 200e6/nbins;
```

```
dt = 1/200e6;


FFT.DJI      = zeros(length(DJI_pulse.pokus)/nbins, nbins);

for k = 1 : 1 : length(DJI_pulse.pokus)/nbins

%     if do_you_want_DC_removed == 'yes'
%
%         DJI_pulse.pokus((k-1)*nbins + 1 : (k-1)*nbins + nbins) = ...
%             DJI_pulse.pokus((k-1)*nbins + 1 : (k-1)*nbins + nbins) - ...
%             mean(DJI_pulse.pokus((k-1)*nbins + 1 : (k-1)*nbins + nbins));
%
%     end

    FFT.DJI(k, :) = ( abs( fftshift( fft(w.*DJI_pulse.pokus( ...
        (k-1)*nbins + 1 : (k-1)*nbins + nbins) ) ) )*dt);


end

figure()
plot(1:1:nbins, 10*log10(FFT.DJI(1, :)));


%% INCOHERENT INTEGRATION / AVERIGING FILTER
num_of_ave = 1;

AVE.FFT.DJI = FFT.DJI(1:num_of_ave, :)/num_of_ave;

%AVE.FFT.DJI = sum(FFT.DJI(1:num_of_ave, :), 1)/num_of_ave;
%figure()
%plot(1:1:nbins, FFT.DJI(1, :));
%hold on
%plot(AVE.FFT.DJI)


%% ADAPTIVE THRESHOLD + DETECTION
% mind the scaling factor, which is necessary to "lift the threshold up" a
% little bit. Also, the scaling factor will differ from CFAR
% computation performed on logarithmed spectrum and CFAR computaiton
% performed on linear spectrum
%
% The scaling factor also needs to be decereased for thersholding a signal
% with low SNR.
%
% CA-CFAR COMPARISON

ww          = [4, 8, 16];
gr          = [1, 2, 3];
K           = 12; % Kth largest sample from neighbouring samples
scaling     = 2.9;
select_cfar = 1;


figure('units','normalized','outerposition',[0 0 1 1])
%figure()
for k = 1 : 1 : 3

    for kk = 1 : 1 : 3

        DET.AVE.FFT.DJI = cfarthres((AVE.FFT.DJI), ww(k), gr(kk),...
```

```
            K, scaling, select_cfar);

        subplot(3,3, kk + ((k-1) * 3))
            plot(10*log10(AVE.FFT.DJI))
                hold on
            plot(10*log10(DET.AVE.FFT.DJI))
                xlim([0 512])
                title(sprintf('CA,neigh=%d,gr=%d,scl=%.3f', ww(k), ...
                    gr(kk), scaling))
                xlabel('frequency bin')
                ylabel('log-PSD')

    end

end
print('DJI_pulse_CA_CFAR_detection', '-depsc');
pause

% % OS-CFAR COMPARISON - ADAPTIVE THRESHOLD + DETECTION

K           = [5 11 26]; % Kth largest sample from neighbouring samples
select_cfar        = 2;

figure('units','normalized','outerposition',[0 0 1 1])

for k = 1 : 1 : 3

    for kk = 1 : 1 : 3

        DET.AVE.FFT.DJI = cfarthres(AVE.FFT.DJI, ww(k), gr(kk), K(k), ...
            scaling, select_cfar);

        subplot(3,3, kk + ((k-1) * 3))
            plot(10*log10(AVE.FFT.DJI))
                hold on
            plot(10*log10(DET.AVE.FFT.DJI))
                xlim([0 512])
                title(sprintf('OS,neigh=%d,gr=%d,scl=%.3f,K=%d', ww(k), ...
                    gr(kk), scaling, K(k)))
                xlabel('frequency bin')
                ylabel('PSD')

    end

end
print('DJI_pulse_OS_CFAR_detection', '-depsc');
pause


% MED-CFAR COMPARISON - ADAPTIVE THRESHOLD + DETECTION

select_cfar        = 3;
figure('units','normalized','outerposition',[0 0 1 1])

for k = 1 : 1 : 3

    for kk = 1 : 1 : 3

        DET.AVE.FFT.DJI = cfarthres(AVE.FFT.DJI, ww(k), gr(kk), K, ...
            scaling, select_cfar);

        subplot(3,3, kk + ((k-1) * 3))
```

```
            plot(10*log10(AVE.FFT.DJI))
                hold on
            plot(10*log10(DET.AVE.FFT.DJI))
                xlim([0 512])
                title(sprintf('MED,neigh=%d,gr=%d,scl=%.3f', ww(k),...
                    gr(kk), scaling))
                xlabel('frequency bin')
                ylabel('PSD')

    end

end
print('DJI_pulse_MED_CFAR_detection', '-depsc');
pause

% CAGO-CFAR COMPARISON - ADAPTIVE THRESHOLD + DETECTION

select_cfar        = 4;
figure('units','normalized','outerposition',[0 0 1 1])

for k = 1 : 1 : 3

    for kk = 1 : 1 : 3

        DET.AVE.FFT.DJI = cfarthres(AVE.FFT.DJI, ww(k), gr(kk), K, ...
            scaling, select_cfar);

        subplot(3,3, kk + ((k-1) * 3))
            plot(10*log10(AVE.FFT.DJI))
                hold on
            plot(10*log10(DET.AVE.FFT.DJI))
                xlim([0 512])
                title(sprintf('CAGO,neigh=%d,gr=%d,scl=%.3f', ww(k), ...
                    gr(kk), scaling))
                xlabel('frequency bin')
                ylabel('PSD')

    end

end
print('DJI_pulse_CAGO_CFAR_detection', '-depsc');
pause

% CASO-CFAR COMPARISON - ADAPTIVE THRESHOLD + DETECTION

select_cfar        = 5;
figure('units','normalized','outerposition',[0 0 1 1])

for k = 1 : 1 : 3

    for kk = 1 : 1 : 3

        DET.AVE.FFT.DJI = cfarthres(AVE.FFT.DJI, ww(k), gr(kk), K,...
            scaling, select_cfar);

        subplot(3,3, kk + ((k-1) * 3))
            plot(10*log10(AVE.FFT.DJI))
                hold on
            plot(10*log10(DET.AVE.FFT.DJI))
                xlim([0 512])
                title(sprintf('CASO,neigh=%d,gr=%d,scl=%.3f', ww(k),...
                    gr(kk), scaling))
```

```
                xlabel('frequency bin')
                ylabel('PSD')

    end

end
print('DJI_pulse_CASO_CFAR_detection', '-depsc');

%%
% comparison of all
ww          = [16];
gr          = [3];
K           = 12; % Kth largest sample from neighbouring samples
scaling     = 2.9;
LINEWIDTH   = 1.75;
FONTSIZE    = 12;
BW          = 100e6;

CA.AVE.FFT.DJI   = cfarthres(AVE.FFT.DJI, ww, gr, K, scaling, 1);
OS.AVE.FFT.DJI   = cfarthres(AVE.FFT.DJI, ww, gr, K, scaling, 2);
MED.AVE.FFT.DJI  = cfarthres(AVE.FFT.DJI, ww, gr, K, scaling, 3);
CAGO.AVE.FFT.DJI = cfarthres(AVE.FFT.DJI, ww, gr, K, scaling, 4);
CASO.AVE.FFT.DJI = cfarthres(AVE.FFT.DJI, ww, gr, K, scaling, 5);

figure('units','normalized','outerposition',[0 0 1 1])

plot(10*log10(AVE.FFT.DJI), 'Linewidth', LINEWIDTH-0.5)
    hold on
plot(10*log10(CA.AVE.FFT.DJI), 'Linewidth', LINEWIDTH)
    hold on
plot(10*log10(OS.AVE.FFT.DJI), 'Linewidth', LINEWIDTH)
    hold on
plot(10*log10(MED.AVE.FFT.DJI), 'Linewidth', LINEWIDTH)
    hold on
plot(10*log10(CAGO.AVE.FFT.DJI), 'Linewidth', LINEWIDTH)
    hold on
plot(10*log10(CASO.AVE.FFT.DJI), 'Linewidth', LINEWIDTH)
     hold on
ar = area(1:1:512, (10*log10(AVE.FFT.DJI)), -90,   ...
        'FaceColor', [0 0.4470 0.7410], ...
        'LineStyle', 'none');
    alpha(ar,.11)
    title(sprintf('CFAR comparison,neigh=%d,gr=%d,alpha=%.3f', ww, gr, scaling))
    xlim([0 512])
    ylabel('log-magnitude', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    xlabel('frequency [MHz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
    legend('Rx signal sepctrum','CA','OS','MED','CAGO','CASO', 'spectrum', ...
    'Location', 'southeast')
    xticks([0 nbins/4 nbins/2 (nbins/2+nbins/4) nbins]) % set xaxis ticks
    xt  = get(gca, 'XTick');                            % relabel xticks
    xt2 = [0  BW/4 BW/2 BW/2+BW/4 BW]/1e6;
    set(gca, 'XTick', xt, 'XTickLabel', xt2);
    %xtickangle(30)                                    % rotate xticks
    print('DJI_pulse_CFAR_compare_detection', '-depsc');
```

A.6 Detection in Measured UAV Signal Record

```
%% FHSS measurement CFAR detection
%   MATLAB R2017a
%   running on Lenovo X230
```

```
%   Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz
%   RAM: 12288 MB
%
%   Reading from the LabVIEW binary file containgn USRP record of FPGA
%   FFTed data
%
%   REQUIREMENTS:
%   function: LVbin2matlabdata.m
%             cfarthres
%
%   WARNING: Make sure you have enough RAM
%
%   RECOMMENDATION: run the script section by section

close all; clc; clear;

%% COLOURS

LINEWIDTH       = 1;
LINEWIDTH_AXIS  = 1;
BLUE_DEF        = [0 0.4470 0.7410];
ORANGE_DEF      = [0.8500 0.3250 0.0980];
YELLOW_DEF      = [0.9290 0.6940 0.1250];
PURPLE_DEF      = [0.4940 0.1840 0.5560];
GREEN_DEF       = [0.4660 0.6740 0.1880];
AQUA_DEF        = [0.3010 0.7450 0.9330];
RED_DEF         = [0.6350 0.0780 0.1840];
FONTSIZE        = 12;

%% loading data from the binary file

sample_rate = 100e6;    % sampling frequency set in LabVIEW
nsamples    = 100024;   % number of samples, which were transfered between
                        % the FPGAs FIFO memory and comp memory at one chunk

filename    = 'meas_20_100MHz_DJI6_fixed_fetching.bin';
% use files 'meas_21_100MHz_BT_161018.bin'
% or 'meas_20_100MHz_DJI6_fixed_fetching.bin'

[recorded_data, recording_time] = LVbin2matlabdata(filename, nsamples, sample_rate);


%% reshaping received data array into one vector of consecutive samples
% also convert in double for futuer Matlab computations
%recorded_data = wave_data;

rec_samples = double(reshape ...
    (recorded_data, 1, size(recorded_data, 1) * size(recorded_data, 2)));

disp(sprintf('YOUR RECORDED SIGNAL IS %d SECONDS LONG', recording_time));


%% emptying RAM

clear recorded_data

%% computing the FFT and plot the results

BW            = 100e6;          % signal bandwidth
fs            = 2*BW;            % fs  = sample_rate

winlist = {@barthannwin, ...     % 1
```

```
            @bartlett, ...        % 2
            @blackman, ...        % 3
            @blackmanharris, ...  % 4
            @bohmanwin, ...       % 5
            @chebwin, ...         % 6
            @flattopwin, ...      % 7
            @gausswin, ...        % 8
            @hamming, ...         % 9
            @hann, ...            % 10
            @kaiser, ...          % 11
            @nuttallwin, ...      % 12
            @parzenwin, ...       % 13
            @rectwin, ...         % 14
            @tukeywin, ...        % 15
            @triang};             % 16
wf      = 10;                     % select the window from the list
nbins   = 512;                    % size of the taken DFT
int_T   = 10;                     % integration period = num of averaged DFT snapshopts
B       = rectwin(int_T);         % weightening window, keep it at rectwin,

SNR     = -14;
%SNR     = 50;

nrows   = 10e3;
chunk   = nbins * nrows;

df = 200e6/nbins;
dt = 1/200e6;

ww          = 16;
gr          = 3;
K           = 12; % Kth largest sample from neighbouring samples
scaling     = 2.3;
select_cfar = 1;

fprintf('PLOTTING THE SIGNAL TIME INTERVAL LENGTH: %d seconds\n', chunk*1/BW)
fprintf('TIME RESOLUTION: %d seconds\n', chunk*1/BW)
fprintf('FREQUENCY RESOLUTION: %d Hz\n', BW/nbins )

win = window(winlist{wf}, nbins)';

s_n                = zeros(1, chunk);
FFT.s_n            = zeros(nrows, nbins);
INT.FFT.s_n        = zeros(nrows, nbins);
THR.INT.FFT.s_n    = zeros(nrows, nbins);
DET.THR.INT.FFT.s_n = zeros(nrows, nbins);


figure('units','normalized','outerposition',[0 0 1 1])
for k = 18 : 1 : floor(length(rec_samples)/chunk)

    fprintf('TIME INTERVAL CHUNK %d of %d\n',k,floor(length(rec_samples)/chunk))

    % adding noise
    s_n = awgn(rec_samples( (k-1)*chunk + 1 : k*chunk), SNR, 'measured');

    disp('DFT IN PROGRESS ...')

    for kk = 1 : 1 : nrows

        windowed_st   = win.*s_n((kk-1)*nbins + 1 : kk*nbins );
        FFT.s_n(kk, :) = ( abs( fftshift(fft( (windowed_st), nbins))*dt));
```

```
end

disp('INTEGRATION IN PROGRESS ...')

for kkk = 1 : 1 : nbins


      INT.FFT.s_n(:, kkk) = filter(B, 1, (FFT.s_n(:,kkk)))/int_T;

end

% avoid integration initialization
INT.FFT.s_n(1:int_T, :) = FFT.s_n(1:int_T, :);

% computing threshold
THR.INT.FFT.s_n = cfarthres(INT.FFT.s_n, ww, gr,...
        K, scaling, select_cfar);

% detection
DET.THR.INT.FFT.s_n = INT.FFT.s_n > THR.INT.FFT.s_n;


subplot(121)
imagesc(10*log10(FFT.s_n))
colorbar
colormap('jet')
xticks([0 nbins/4 nbins/2 (nbins/2+nbins/4) nbins]) % set xaxis ticks
xt  = get(gca, 'XTick');                            % relabel xticks
xt2 = [0  BW/4 BW/2 BW/2+BW/4 BW]/1e6;
set(gca, 'XTick', xt, 'XTickLabel', xt2);
%xtickangle(30)                                     % rotate xticks
yt  = get(gca, 'YTick');                            % set yaxis ticks
yt2 = (yt*nbins/1e2);                               % relabel yticks
set(gca, 'YTick', yt, 'YTickLabel', yt2);
ylabel('time [us]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
xlabel('frequency [MHz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
hcb = colorbar;
title(hcb, 'log-magnitude')
%print('DJI_spec_512_hann_compare', '-depsc')

subplot(122)
imagesc((DET.THR.INT.FFT.s_n))
colorbar
colormap('jet');
%colormap('bone')
xticks([0 nbins/4 nbins/2 (nbins/2+nbins/4) nbins]) % set xaxis ticks
xt  = get(gca, 'XTick');                            % relabel xticks
xt2 = [0  BW/4 BW/2 BW/2+BW/4 BW]/1e6;
set(gca, 'XTick', xt, 'XTickLabel', xt2);
%xtickangle(30)                                     % rotate xticks
yt  = get(gca, 'YTick');                            % set yaxis ticks
yt2 = (yt*nbins/1e2);                               % relabel yticks
set(gca, 'YTick', yt, 'YTickLabel', yt2);
ylabel('time [us]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
xlabel('frequency [MHz]', 'FontSize', FONTSIZE, 'FontWeight', 'bold')
hcb = colorbar;
title(hcb, 'log-magnitude')

%print('DJI_spec_512_hann_integrated_10_LOW_SNR_THR_DET', '-depsc')

disp('PRESS A KEY TO CONTINUE TO THE NEXT CHUNK')
```

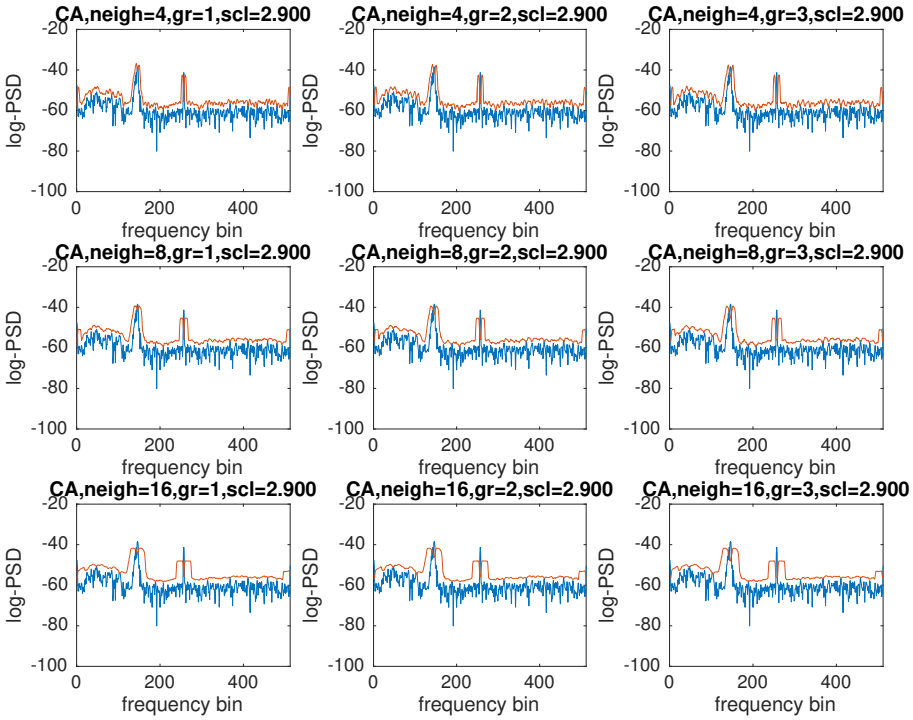
```
    pause;


end
```

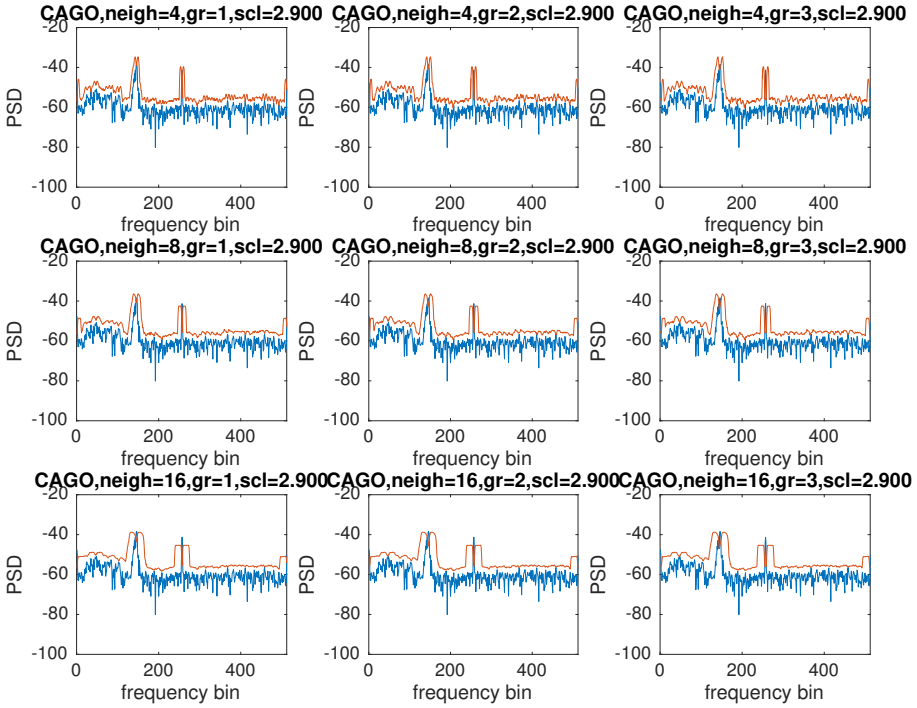
Appendix B

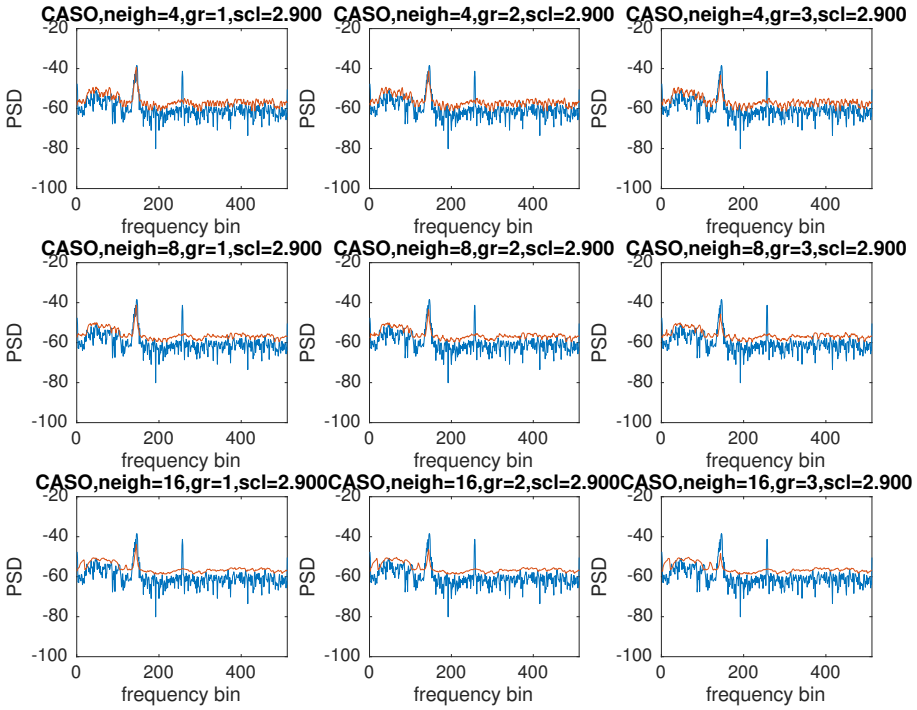
Appendix: CFAR Comparison

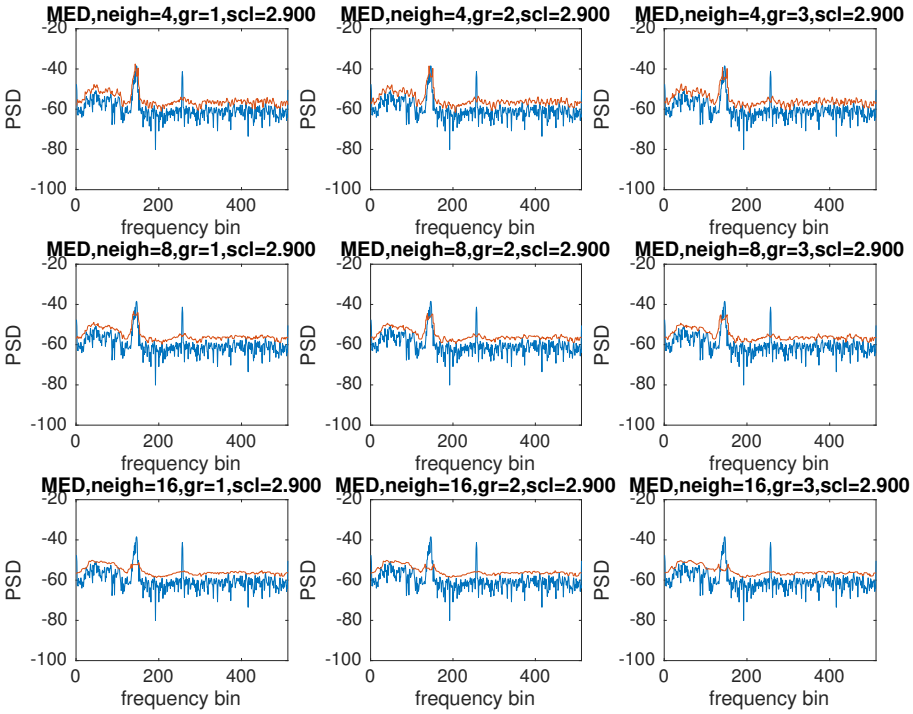
B.1 Comparison of CFAR Processors

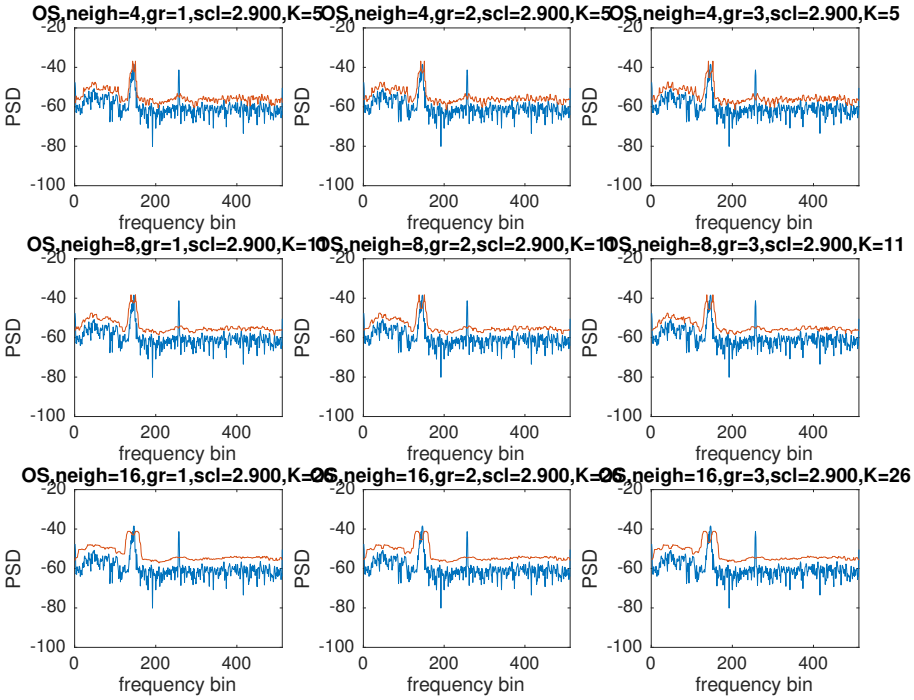
In this section, we present a comparison of CFAR processors described in chapter 7. The processors are also demonstrated with several different settings like the size of the neighbouring, or guarding windows, eventually the order selecting K factor for the OS-CFAR. The scaling factor α remains the same for all graphs. The information are stated inside the figure corresponding to each graph.


CA,neigh=4,gr=1,scl=2.900
CA,neigh=4,gr=2,scl=2.900
CA,neigh=4,gr=3,scl=2.900
CA,neigh=8,gr=1,scl=2.900
CA,neigh=8,gr=2,scl=2.900
CA,neigh=8,gr=3,scl=2.900
CA,neigh=16,gr=1,scl=2.900
CA,neigh=16,gr=2,scl=2.900
CA,neigh=16,gr=3,scl=2.900
log-PSD
frequency bin
-20
-40
-60
-80
-100
0
200
400


CAGO,neigh=4,gr=1,scl=2.900
CAGO,neigh=4,gr=2,scl=2.900
CAGO,neigh=4,gr=3,scl=2.900
CAGO,neigh=8,gr=1,scl=2.900
CAGO,neigh=8,gr=2,scl=2.900
CAGO,neigh=8,gr=3,scl=2.900
CAGO,neigh=16,gr=1,scl=2.900
CAGO,neigh=16,gr=2,scl=2.900
CAGO,neigh=16,gr=3,scl=2.900
PSD
frequency bin


CASO,neigh=4,gr=1,scl=2.900
CASO,neigh=4,gr=2,scl=2.900
CASO,neigh=4,gr=3,scl=2.900
CASO,neigh=8,gr=1,scl=2.900
CASO,neigh=8,gr=2,scl=2.900
CASO,neigh=8,gr=3,scl=2.900
CASO,neigh=16,gr=1,scl=2.900
CASO,neigh=16,gr=2,scl=2.900
CASO,neigh=16,gr=3,scl=2.900
PSD
frequency bin


MED,neigh=4,gr=1,scl=2.900
MED,neigh=4,gr=2,scl=2.900
MED,neigh=4,gr=3,scl=2.900
MED,neigh=8,gr=1,scl=2.900
MED,neigh=8,gr=2,scl=2.900
MED,neigh=8,gr=3,scl=2.900
MED,neigh=16,gr=1,scl=2.900
MED,neigh=16,gr=2,scl=2.900
MED,neigh=16,gr=3,scl=2.900
PSD
frequency bin


OS,neigh=4,gr=1,scl=2.900,K=5
OS,neigh=4,gr=2,scl=2.900,K=5
OS,neigh=4,gr=3,scl=2.900,K=5
OS,neigh=8,gr=1,scl=2.900,K=11
OS,neigh=8,gr=2,scl=2.900,K=11
OS,neigh=8,gr=3,scl=2.900,K=11
OS,neigh=16,gr=1,scl=2.900,K=26
OS,neigh=16,gr=2,scl=2.900,K=26
OS,neigh=16,gr=3,scl=2.900,K=26
PSD
frequency bin