CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Radio Engineering

Master's Thesis

Implementation of Simple WPNC System on the Experimental Transceiver Network Testbed

Jozef Lukáč
Communication and Signal Processing, Open Electronic Systems
lukacjo1@fel.cvut.cz

May 2019
Supervisor: prof. Ing. Jan Sýkora, CSc.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Lukáč**	Jméno: **Jozef**	Osobní číslo: **434697**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra radioelektroniky**		
Studijní program:	**Otevřené elektronické systémy**		
Studijní obor:	**Komunikace a zpracování signálu**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Implementace jednoduchého WPNC systému v experimentální síti laboratorních transceiverů

Název diplomové práce anglicky:

Implementation of Simple WPNC System on the Experimental Transceiver Network Testbed

Pokyny pro vypracování:

The student will get acquainted with the fundamentals of WPNC (Wireless Physical Layer Network Coding) with isomorphic layered NCM, channel estimation algorithms for hierarchical MAC channel, and with the laboratory experimental Ettus-USRP based transceiver network testbed. The work goal is to implement a simple end-to-end WPNC system including layered NCM, H-decoding, H-BC stage, and H-MAC/H-BC channel estimation. The scenario should include at least butterfly network topology with PSK modulated sources using outer state-of-the-art code (e.g. LDPC). An optional goal may include an extension for more than 2 source nodes and/or some form of encapsulated network. The system should first be implement by a computer simulation which will later serve as a benchmark reference for the over-the-air experiments.

Seznam doporučené literatury:

[1] Jan Sykora, Alister Burr: Wireless Physical Layer Network Coding, Cambridge University Press, 2018
[2] dokumentace Ettus USRP

Jméno a pracoviště vedoucí(ho) diplomové práce:

prof. Ing. Jan Sýkora, CSc., katedra radioelektroniky FEL

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

Datum zadání diplomové práce: **28.01.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

prof. Ing. Jan Sýkora, CSc.	prof. Mgr. Petr Páta, Ph.D.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	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

Acknowledgement

I would like to thank to prof. Ing. Jan Sýkora, CSc. for giving me the initial concept of the whole thesis, for his leadership, consultations and his time. Next, I thank to my parents and whole my family.

Declaration

Abstrakt / Abstract

V této práci vyvíjíme simulaci bezdrátové sítě, konkrétně sítě využívající principy WPNC. Na začátku jsou odvozeny estimátory pro H-MAC a H-BC kanály. Následuje implementace odvozených estimátorů v MATLABu společně s implementací pomocných objektů a funkcí. Dále prezentujeme softwarová rádia a jejich vlastnosti. Základní, čistě počítačová simulace přenosu one-to-one je vytvořená v MATLABu, za ní následuje simulace využívající softwarová rádia. Na konci jsou prezentovány přijaté obrázky. Z matematických metod byla použita Monte Carlo Integrace a odvození s pomocí skalárního součinu. Z programovacích technik bylo použito Objektově orientované a funcionální programování s aplikací vektorových operací. Jedním z nejvýznamnějších výsledků této práce je úspěšný přenos a dekódování několika Hierarchických rámců v H-MAC kanálu v simulaci se softwarovými rádii.

Klíčová slova: WPNC, Softwarové rádio, MATLAB simulace

In the thesis we develop a simulation of a wireless network. Specifically, a network that uses WPNC principles. In the beginning, there are estimators for H-MAC and H-BC derived. A MATLAB implementation of the estimators follows together with implementation of auxiliary objects and functions. Next, software radios are presented and their properties are examined. A basic MATLAB pure-computer simulation of a one-to-one transmission is developed followed by over-the-air simulation. In the end, resulted received pictures are examined. From the math methods, Monte Carlo Integration and derivations with dot product were used. From the programming methods, Object Oriented and Functional Programming were used with vectorized operations. One of the significant results of the work is a successful transmission, reception and decoding of several Hierarchical frames in H-MAC stage in the over-the-air simulation.

Keywords: WPNC, Software Radio, MATLAB simulation

/ Contents

Tables / Figures

Chapter 1
Introduction

Wireless Physical layer Network Coding (WPNC) is relatively a new concept, emerging around early 2000's (see e.g. [1]).

One of the key concepts of WPNC is using interference of two or more signals for increasing capacity of a wireless communication channel. Wireless channel implicitly enables and cannot easily prevent shared access. Transmitted signals in one stage are not orthogonal, they are transmitted at the same frequency and at the same time-slot. Properly prepared and maintained interference is not perceived as harmful.

In Multiple Access Channel (MAC) phase, several transmitters are transmitting concurrently. Signals are superimposed at physical layer, electromagnetic waves interference occurs. A receiver senses the mixture of signals. From the received signal then Rx tries to determine, resp. to compute a function of original signals. The output of that function is transmitted at a next stage.

The aim of the thesis is to demonstrate basic principles of WPNC in some simple network topologies by simulations and using software radios (SRs) for an over-the-air transmission.

All algorithms were developed in MATLAB program environment.

1.1 Basic definitions and concepts

In the section we will introduce some notions of WPNC necessary for the following work. There will be made an effort to briefly describe required knowledge also for a "hands-off" reader. The sections draws knowledge mainly from the [2].

In a general WPNC network there are source-nodes, destination-nodes and relay-nodes. Let's now focus to relay-nodes strategies. We distinguish 3 types of strategies:

- Amplify and Forward (AF). The simplest strategy. The relay just amplifies received signal and retransmits it. All the processing is left to destination-nodes. One of drawbacks is the noise-amplification (together with received signal).
- Joint Decode and Forward (JDF). Each source in MAC stage is decoded separately. Then a network code function is applied to the decoded symbols, and the result is broadcasted to destinations. An advantage is the possibility to choose an arbitrary function.
- Hierarchical Decode and Forward (HDF). The relay decodes network-coded function directly from the received signal. It does not try to decode each source separately, it attempts to decode the output of a given network code function, as if the result signal was transmitted from one source. Such a signal we will denote as hierarchical. Hence the name Hierarchical Decoding.

In the work, we will aim our attention at HDF; it assumes so called Network Coded Modulation (NCM).

Definition 1.1. *Network Coded Modulation* is a channel encoding taking into account the network structure and channel-model. Literally, code is applied by the network according to the channel-model. Every single source particularly contributes to the overall code. The Hierarchical signal (H-signal) is received in a superposition with other NCM signals in the same stage. E.g. for a channel model: $x = s_A + s_B$. Where x is a received symbol; s_A, s_B are BPSK symbols (± 1) from two source nodes: A, B. A receiver senses Hierarchical-symbols (H-symbols): -2, 0, 2.

In the following text we will hold a rule for naming:

- b_i – will be raw binary data from the source i. It will be one symbol, a data-word or a whole message depending on a context.
- c_i – will be encoded binary data by a code $\mathcal{C}_i$. So $c_i = \mathcal{C}_i(b_i)$.
- s_i – will be constellation point(s) resulting from the c_i encoded data. The one to one mapping will be marked as $\mathcal{A}_i$: $s_i = \mathcal{A}_i(c_i)$.

As in the classical communication networks also in networks using WPNC there is defined a certain layered approach. (cf. ISO/OSI model [1]) – the Layered NCM.

Definition 1.2. *Layered NCM* consists of (1) outer codebooks $\mathcal{C}_i$ with discrete encoded symbols $c_i = \mathcal{C}_i(b_i)$ and (2) inner constellation space symbol one-to-one mappers $s_i = \mathcal{A}_i(c_i)$. We define Hierarchical Network Code (HNC) mapping (i.e. a mapping for NCM) on *both* data $\bar{b} = \{b_i\}_i$ (marking for a set of raw binary data of all source nodes) and outer layer encoded symbols $\tilde{c} = \{c_i\}_i$. I.e. $b = \chi(\bar{b}) \equiv \chi(b_1, b_2, \ldots, b_n)$ and $c = \chi_c(\tilde{c}) \equiv \chi_c(c_1, c_2, \ldots, c_n)$. The H-constellation associated with Layered NCM is the one related to the outer layer encoded symbols

$$\mathcal{U}(c) = \left\{u : u = u\left(\{s_i(c_i)\}_{i\in\{1,2,\ldots,n\}}, \tilde{h}\right) | c = \chi_i(\tilde{c})\right\}, \tag{1}$$

where $\tilde{h}$ is a vector channel parameter: each element for one Tx-to-Rx channel. We also define a product component code $\tilde{C} = \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$. $\tilde{c} = \tilde{C}(\bar{b})$.

One further specification of Layered NCM is Isomorphic Layered NCM.

Definition 1.3. *Isomorphic Layered NCM*is Layered NCM consisting of outer codes $\mathcal{C}_i$, $c_i = \mathcal{C}_i(b_i)$, and HNC data and code symbol maps $b = \chi(\bar{b})$, $c = \chi_c(\tilde{c})$ such that there exists a valid one-to-one equivalent *isomorphic* hierarchical codebook (IH-codebook) $\mathcal{C}$, such that $c = \mathcal{C}(b)$, i.e.

$$\forall \tilde{b} : c = \chi_c\left(\tilde{\mathcal{C}}(\tilde{b})\right) = \mathcal{C}\left(\chi(\tilde{b})\right). \tag{2}$$

In our work we will use the same linear code $\mathcal{C}_{\text{com}}$ [2] for every source node, specifically LDPC code and HNC mapping i.e. χ and χ_c will be XOR function. So the condition for Isomorphic Layered NCM is automatically fulfilled. In our case:

$$\begin{aligned}
\tilde{\mathcal{C}} &= \mathcal{C}_{\text{com}} \times \mathcal{C}_{\text{com}} \times \cdots \times \mathcal{C}_{\text{com}},\\
\mathcal{C} &= \mathcal{C}_{\text{com}},\\
\tilde{b} &= (b_1, b_2, \ldots, b_n),\\
\chi_c &= \chi = \text{XOR}
\end{aligned}$$

$$\begin{aligned}
c = \chi_c(\tilde{\mathcal{C}}(\tilde{b})) &\equiv \chi_c(\mathcal{C}_{\text{com}}(b_1), \mathcal{C}_{\text{com}}(b_2), \ldots, \mathcal{C}_{\text{com}}(b_n)),\\
&= \mathcal{C}_{\text{com}}(b_1) \oplus \mathcal{C}_{\text{com}}(b_2) \oplus \cdots \oplus \mathcal{C}_{\text{com}}(b_n),\\
&\overset{(a)}{=} \mathcal{C}_{\text{com}}(b_1 \oplus b_2 \oplus \cdots \oplus b_n)\\
&= \mathcal{C}_{\text{com}}(\chi(\tilde{b})) = \mathcal{C}(\chi(\tilde{b})).
\end{aligned} \tag{3}$$

[1] `https://en.wikipedia.org/wiki/OSI_model`
[2] com – common

(a) – linearity of XOR function.

In a WPNC network a MAC and Broadcast Channel BC stages have an adjective Hierarchical H-.

1.2 Examined topology

The network topology we have examined is called butterfly network 1.1.

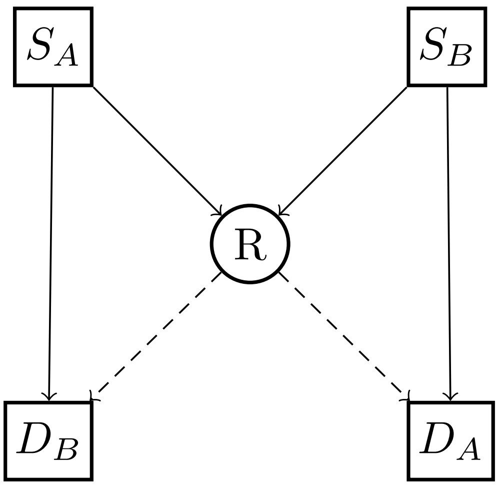

Figure 1.1. Butterfly network topology.

It consists of two sources: S_A, S_B, two destinations D_A, D_B and one relay-node R. The communication schedule is following:

1. Stage 1 (H-MAC): S_A and S_B transmits, R, D_A and D_B receives. R receives H-signal whereas D_A, D_B receive an ordinary signal from a SISO channel.
2. Stage 2 (H-BC): R decodes H-symbols and transmits them to nodes D_A and D_B. The destinations then can decode messages from both sources: S_A and S_B.

Now, we will explain technicalities involved. At the raw-binary data view:

In MAC stage b_A and b_B are transmitted. HNC mapping is XOR function, so relay receives: $b = b_A \oplus b_B$, D_A receives b_A; D_B receives b_B. In BC stage, R transmits b to the D_A, D_B. D_A has in MAC stage received b_A and for obtaining b_B executes following processing: $b_B = b_A \oplus b$. Similarly D_B has in MAC stage received b_B and for obtaining b_A performs: $b_A = b_B \oplus b$.

Because we have used linear code and XOR function as HNC function. The communication is isomorphic layered NCM 1.3. And similar derivation hold also for encoded data: c_A, c_B.

At the PHY layer we assume an AWGN channel with attenuation and phase rotation: $x = h_i \cdot s_i + w$. So for the relay we have

$$x = h_A \cdot s_A + h_B \cdot s_B + w = h_A(s_A + h \cdot s_B) + w. \tag{4}$$

Where $h = h_B/h_A$ is called *relative attenuation* and h_A is called *common attenuation*. We can see the relay as receiving H-symbols: $s_A + h \cdot s_B$ in the ordinary AWGN channel with attenuation. We see, H-symbols depend on relative channel attenuation. From the H-symbols the Relay tries to decode output of the XOR function hypothetically performed in background on c_A and c_B data; this is called H-decoding.

In the pictures 1.2 and 1.3 we can see H-symbols for 2 different relative channel attenuations.

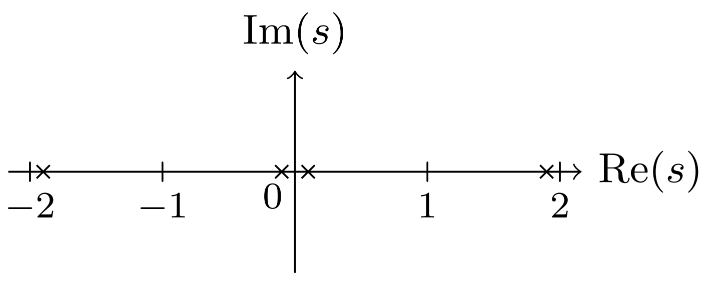

Figure 1.2. Example of H-symbols for relative attenuation $h = 0.9$.

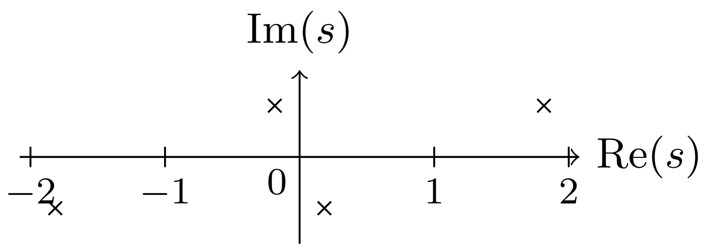

Figure 1.3. Example of H-symbols for relative attenuation $h = 0.9 \exp(\mathrm{j}25/180 \cdot \pi)$.

Chapter 2
Transmitter and Receiver composition

The simulation is developed in the MATLAB environment, as a two separate processes running MATLAB. Every communication stage is simulated as a transmitter(s) in one process and receiver(s) in the second and vice versa. In the figure 2.1 we can see a basic transmitter composition, in the figure 2.2 there is a basic schema of a receiver. In this section we will explain the blocks that are included in these schema.

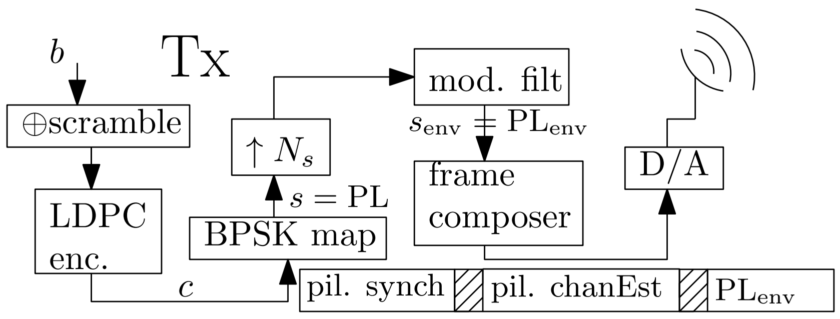

Figure 2.1. A transmitter composition.

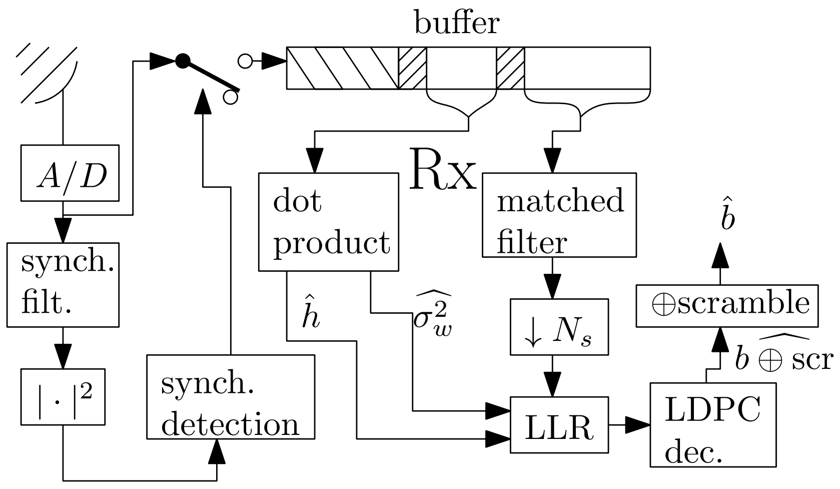

Figure 2.2. A receiver composition.

Two separate approaches are implemented:

- Pure computer simulation. The communication channel is implemented by means of a shared file. The processes communicate through the shared file.
- Over-the-air simulation. The processes use an interface with Software radios in MATLAB. Interfaces are implemented by MATLAB System object [1].

[1] `https://ch.mathworks.com/help/matlab/matlab_prog/what-are-system-objects.html`

The program is designed such that a conversion from pure computer simulation to over-the-air simulation requires minimum changes in the code.

Both kinds of simulation should enable a mode (a) for a numerical evaluation of performance and (b) for a real-time visual simulation. Black&White pictures are being sent in a loop and received images are being drawn if required.

2.1 Frame composition

We will consider communication using frames. Further, because the transmission should run on Software radios with its antennas close to each other, i.e. there is implicitly a radio visibility between each pair of antennas, we have to divide H-MAC stage to three phases:

- Phase 1: S_A, S_B transmit; R receives.
- Phase 2: S_A transmits, D_A receives.
- Phase 3: S_B transmits, D_B receives.

For phase 2 and 3 of H-MAC stage we have chosen a frame according to the picture 2.3. The frame is composed of a pilot signal and Payload. Pilot signal is divided to two parts: first for synchronization and second for channel-parameters estimation.

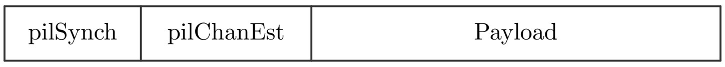

Figure 2.3. A frame composition for the transmission of source S_A and S_B in Phase 2 and Phase 3 of H-MAC stage.

The frame structure 2.3 has been chosen also for BC stage. Between every pair of subsequent parts of the frame, several zero-samples have been inserted there.

For phase 1 we have chosen a frame according to the picture 2.4. Pilot signals of both sources are orthogonal in time.

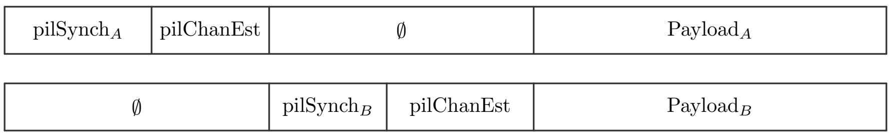

Figure 2.4. A frame composition for the transmission of source S_A and S_B in Phase 1 of H-MAC stage.

Synchronization part and channel estimation part of the pilot signal consists of a pseudorandom (PN) BPSK sequence modulated by a short rectangular (REC) pulse of order (2 or 3 samples per symbol). Payload signal is prepared as follows:

1. Raw binary data b_i are split into frames of length suitable for LDPC encoding. LDPC code is chosen according to the DVB-S2 standard [3]. There is a codeword length n_{ldpc} chosen first. It is either short $n_{\text{ldpc}} = 16200$ bits or long $n_{\text{ldpc}} = 64800$ bits. Then, according to a code rate the dataword-length is computed.
2. A spread-sequence, is applied. Raw data are first scrambled by XORing them with a spread-sequence. The spread-sequence is also chosen from the DVB-S2 standard [3](pg. 21 BaseBand scrambling).
3. LDPC encoding follows.
4. BPSK constellation space mapping: $(b_i - 0.5)\cdot 2$, (i.e. $0 \to -1$, $1 \to 1$).
5. Zero-padding and modulation using RRC pulse.

2.2 Synchronization sequence

Synchronization sequence is used to locate the frame in a stream of data. We will build ML estimator for a *delay* of the frame in the complex AWGN (CWGN) channel model for continuous time: $x(t) = \gamma \exp(\mathrm{j}\varphi)s(t-\tau) + w(t)$. $h := \gamma \exp(\mathrm{j}\varphi)$ – channel attenuation.

Likelihood function of the channel model is:

$$p(x(t)|\gamma, \varphi, \tau) = \alpha \exp\left\{-1/\sigma_w^2 \|x(t) - \gamma e^{\mathrm{j}\varphi} s(t-\tau)\|^2\right\}.$$

α is an uninteresting normalization factor. Procedure for the obtaining ML estimator for the delay τ is following: (1) marginalize channel likelihood function over phase φ, then (2) find its arg max w.r.t. τ. A note: Dot product is defined as: $\langle f(t), g(t)\rangle = \int_{\mathbb{R}} f(t)\cdot g^*(t)\,\mathrm{d}t$. $\|s(t-\tau)\|^2 = \langle s(t-\tau), s(t-\tau)\rangle = \|s(t)\|^2$. However, the dot product will be computed from signal-samples. Because it holds: $\langle x(t), s(t)\rangle_{L^2} = \langle \mathbf{x}, \mathbf{s}\rangle_{\ell^2}$. For base functions: $\mathrm{sinc}(t) = \sin(\pi t)/(\pi t)$.

$$\begin{aligned}
p(x(t)|\gamma, \varphi, \tau) &= \alpha \exp\left\{-1/\sigma_w^2 \left(\|x(t)\|^2 + \gamma^2\|s(t)\|^2\right)\right\} \\
&\quad \cdot \exp\left\{2/\sigma_w^2 \Re\left\{\langle x(t), \gamma e^{\mathrm{j}\varphi} s(t-\tau)\rangle\right\}\right\}. \\
p(x(t)|\gamma, \tau) &= \int_{-\pi}^{\pi} p(x(t)|\gamma, \varphi, \tau) p(\varphi)\,\mathrm{d}\varphi, \\
&= c(\gamma)\int_{-\pi}^{\pi} \exp(2\gamma/\sigma_w^2 \Re\{e^{-\mathrm{j}\varphi}\langle x(t), s(t-\tau)\rangle\})\,\mathrm{d}\varphi, \\
&= |\text{ assign sp } = \langle x(t), s(t-\tau)\rangle \text{ (scalar product)}|, \\
&= c(\gamma)\int_{-\pi}^{\pi} \exp(2\gamma/\sigma_w^2 [\cos\varphi \Re\{\mathrm{sp}\} + \sin\varphi \Im\{\mathrm{sp}\}])\,\mathrm{d}\varphi, \\
&= |\text{assign } A = 2\gamma/\sigma_w^2 \Re\{\mathrm{sp}\}, \quad B = 2\gamma/\sigma_w^2 \Im\{\mathrm{sp}\}|, \\
&= c(\gamma)\int_{-\pi}^{\pi} \exp(A\cos\varphi + B\sin\varphi)\,\mathrm{d}\varphi, \\
&\overset{(a)}{=} c(\gamma)\int_{-\pi}^{\pi} \exp\{\sqrt{A^2+B^2}\cos[\varphi - \arctan(B/A)]\}\,\mathrm{d}\varphi, \\
&\overset{(b)}{=} c(\gamma) I_0(-\sqrt{A^2+B^2}) = c(\gamma) I_0(-2\gamma/\sigma_w^2 |\langle x(t), s(t-\tau)\rangle|). \qquad (1)
\end{aligned}$$

(a) – trigonometric identity applied (5). (b) – property of the modified bessel function of the first kind (4).

So our ML delay-estimator:

$$\begin{aligned}
\hat{\tau} &= \arg\max_{\check{\tau}} p(x(t)|\gamma, \check{\tau}), \\
&= \arg\max_{\check{\tau}} c(\gamma) I_0(-2\gamma/\sigma_w^2\ |\langle x(t), s(t-\check{\tau})\rangle|), \\
&\overset{(a)}{=} \arg\max_{\check{\tau}} |\langle x(t), s(t-\check{\tau})\rangle|. \qquad (2)
\end{aligned}$$

(a) – the function $I_0(x) = I_0(-x) = J_0(-\mathrm{j}x)$ is increasing (4). For a saving of computation power we will decimate sequence of received envelope samples by a factor $\mathrm{Ns}_{\mathrm{pil}}$ – number of samples per symbol in pilot – and correlate it with synch. sequence samples (not whole envelope of synchronization sequence). When we detect frame, we start saving the received signal and after whole frame is stored, i.e in postprocessing part, the whole synchronization signal envelope is used for a precise determination of the frame.

2.3 Channel-state estimators

Now, when we have estimated delay of frame, we will estimate state of channel, i.e. *attenuation* γ and *phase rotation* φ. Consider channel model

$$\mathbf{x} = \gamma e^{j\varphi}\mathbf{s} + \mathbf{w}. \tag{3}$$

Likelihood function for the channel model is

$$p(\mathbf{x}|\gamma,\varphi) = \alpha \exp[-1/\sigma_w^2 \|\mathbf{x} - \gamma e^{j\varphi}\mathbf{s}\|^2] = c(\gamma)\exp[2\gamma/\sigma_w^2 \Re\{e^{-j\varphi}\langle \mathbf{x},\mathbf{s}\rangle\}]. \tag{4}$$

So ML phase estimator is:

$$\begin{aligned}\hat{\varphi} &= \arg\max_{\check{\varphi}} p(\mathbf{x}|\gamma,\check{\varphi}) = \arg\max_{\check{\varphi}} \Re\{e^{-j\check{\varphi}}\langle \mathbf{x},\mathbf{s}\rangle\},\\ &= \arg\max_{\check{\varphi}} \Re\{|\langle \mathbf{x},\mathbf{s}\rangle| e^{j(\arg\langle \mathbf{x},\mathbf{s}\rangle - \check{\varphi}}\},\\ &= \arg\langle \mathbf{x},\mathbf{s}\rangle . \end{aligned} \tag{5}$$

Using phase-estimator we will estimate *channel attenuation* γ:

$$\begin{aligned}\hat{\gamma} &= \arg\max_{\check{\gamma}} \ln p(\mathbf{x}|\check{\gamma},\hat{\varphi}),\\ &= \arg\max_{\check{\gamma}} \frac{-1}{\sigma_w^2}\left(\|\mathbf{x}\|^2 + \check{\gamma}^2\|\mathbf{s}\|^2\right) + \frac{2\check{\gamma}}{\sigma_w^2}|\langle \mathbf{x},\mathbf{s}\rangle|.\\ &= \frac{|\langle \mathbf{x},\mathbf{s}\rangle|}{\|\mathbf{s}\|^2}. \end{aligned} \tag{6}$$

$$\frac{\partial \ln p(\mathbf{x}|\check{\gamma},\hat{\varphi})}{\partial\check{\gamma}} = \frac{-2}{\sigma_w^2}\check{\gamma}\|\mathbf{s}\|^2 + \frac{2}{\sigma_w^2}|\langle \mathbf{x},\mathbf{s}\rangle| \overset{!}{=} 0.$$

And finally, we will need to estimate *noise variance*. Using all the estimators above we will estimate noise realization and from it the variance will be computed. Note: We consider complex noise.

$$\begin{aligned}\mathbf{v} &:= \mathbf{x} - \hat{\gamma}e^{j\hat{\varphi}}\mathbf{s},\\ \hat{\sigma_w^2} &= 2\cdot\mathrm{var}\left[\Re\{\mathbf{v}\}\right]. \end{aligned} \tag{7}$$

2.4 Estimator performance – Cramér-Rao Lower Bound

In this section we will evaluate CRLB for all the three estimators. We should note, that we have derived delay-estimator for continuous time τ, but only samples of received signal are available. So we estimate coefficient k_0 of a discrete channel model: $\mathbf{x}[n] = \gamma e^{j\varphi}\mathbf{s}[n-k_0] + \mathbf{w}$ instead of τ.

CRLB is a lower bound on the variance for any *unbiassed* estimator of a *deterministic* (fixed for all samples of the observation vector) parameter [4]. The only input to the evaluation is a relation how the received signal depends on the examined parameter θ (general unknown parameter) – the channel likelihood function: $p(\mathbf{x}|\theta)$. The CRLB says variance of every estimator is [4]:

$$\mathrm{var}[\hat{\theta}] \geq \left(-\mathrm{E}\left[\frac{\partial^2 \ln p(\mathbf{x}|\theta)}{\partial\theta^2}\right]\right)^{-1} = \left(\mathrm{E}\left[\left(\frac{\partial \ln p(\mathbf{x},\theta)}{\partial\theta}\right)^2\right]\right)^{-1}. \tag{8}$$

For the $\hat{\varphi}$ and $\hat{\gamma}$ estimators we have used the CWGN model (4). For the unified evaluation and comparison consider $\|\mathbf{s}\|^2 = 1$ and $\mathrm{SNR} = \gamma^2/\sigma_w^2$. Further we will use: $\langle \mathbf{x}, \mathbf{s} \rangle = \left\langle \gamma \mathrm{e}^{\mathrm{j}\varphi}\mathbf{s} + \mathbf{w}, \mathbf{s} \right\rangle = \gamma \mathrm{e}^{\mathrm{j}\varphi}\|\mathbf{s}\|^2 + \langle \mathbf{w}, \mathbf{s} \rangle$.

$$\begin{aligned}
\frac{\partial \ln p(\mathbf{x}|\gamma,\varphi)}{\partial \varphi} &= \frac{\partial}{\partial \varphi}\frac{2\gamma}{\sigma^2}\Re\{\mathrm{e}^{\mathrm{j}\varphi}\langle \mathbf{x}, \mathbf{s} \rangle\} = \frac{2\gamma}{\sigma_w^2}\Re\{\mathrm{e}^{\mathrm{j}\varphi}(-\mathrm{j})\langle \mathbf{x}, \mathbf{s} \rangle\},\\
\frac{\partial^2 \ln p(\mathbf{x}|\gamma,\varphi)}{\partial \varphi^2} &= \frac{2\gamma}{\sigma_w^2}\Re\{\mathrm{e}^{-\mathrm{j}\varphi}(-\mathrm{j})^2\langle \mathbf{x}, \mathbf{s} \rangle\}.\\
-\mathrm{E}\left[\frac{\partial^2 \ln p(\mathbf{x}|\gamma,\varphi)}{\partial \varphi^2}\right] &= \frac{2\gamma}{\sigma_w^2}\left(\mathrm{e}^{-\mathrm{j}\varphi}\gamma\mathrm{e}^{\mathrm{j}\varphi}\|\mathbf{s}\|^2 + \Re\{\mathrm{e}^{-\mathrm{j}\varphi}\mathrm{E}\left[\langle \mathbf{w}, \mathbf{s} \rangle\right]\}\right) = \frac{2\gamma^2}{\sigma_w^2} = 2\,\mathrm{SNR}.\\
\mathrm{var}[\hat{\varphi}] &\geq \frac{1}{2\,\mathrm{SNR}}. \qquad (9)
\end{aligned}$$

The CRLB of $\hat{\gamma}$ follows:

$$\begin{aligned}
\ln p(\mathbf{x}|\gamma,\varphi) &= \ln\alpha + \left(\frac{-1}{\sigma_w^2}\right)(\|\mathbf{x}\|^2 + \gamma^2\|\mathbf{s}\|^2) + \frac{2\gamma}{\sigma_w^2}\Re\{\mathrm{e}^{-\mathrm{j}\varphi}\langle \mathbf{x}, \mathbf{s} \rangle\}.\\
\frac{\partial \ln p(\mathbf{x}|\gamma,\varphi)}{\partial \gamma} &= -\frac{2\gamma}{\sigma_w^2}\|\mathbf{s}\|^2 + \frac{2}{\sigma_w^2}\Re\{\mathrm{e}^{-\mathrm{j}\varphi}\langle \mathbf{x}, \mathbf{s} \rangle\}.\\
\frac{\partial^2 \ln p(\mathbf{x}|\gamma,\varphi)}{\partial \gamma^2} &= -\frac{2}{\sigma_w^2}\|\mathbf{s}\|^2.\\
-\mathrm{E}\left[\frac{\partial^2 \ln p(\mathbf{x}|\gamma,\varphi)}{\partial \gamma^2}\right] &= \frac{2}{\sigma_w^2}\|\mathbf{s}\|^2 = \frac{2\gamma^2}{\gamma^2\sigma^2} = \frac{2\,\mathrm{SNR}}{\sigma_w^2}.\\
\mathrm{var}[\hat{\gamma}] &\geq \frac{\gamma^2}{2\,\mathrm{SNR}}. \qquad (10)
\end{aligned}$$

For the computation of CRLB of delay-estimator $\hat{\tau}$ several relations will be necessary. One of them is $\frac{\mathrm{d}}{\mathrm{d}x}I_0(x) = I_1(x)$. Next is $\frac{\mathrm{d}}{\mathrm{d}\tau}|\langle x(t), s(t-\tau)\rangle| = -|\langle x(t), s'(t-\tau)\rangle|\cos(\arg\langle x(t), s(t-\tau)\rangle - \arg\langle x(t), s'(t-\tau)\rangle)$ and the last one relation for computing dot-product of a vectors $a(t)$ and $b'(t)$ in L^2 space using sample-vectors $\mathbf{a}$, $\mathbf{b}$. $\langle a(t), b'(t)\rangle = \sum_{n=-\infty}^{\infty} a_n \sum_{q=-\infty}^{\infty} b_{n-q}^* \frac{(-1)^q}{q}, \quad q \neq 0$. See appendix (7) and (8).

$$\begin{aligned}
p(x(t)|\gamma,\tau) &= c(\gamma) I_0\left(\frac{2\gamma}{\sigma_w^2}|\langle x(t), s(t-\tau)\rangle|\right),\\
\frac{\partial \ln p(x(t)|\gamma,\tau)}{\partial \tau} &= \frac{1}{p(x(t)|\gamma,\tau)}\frac{\partial p(x(t)|\gamma,\tau)}{\partial \tau},\\
&= \frac{c(\gamma) I_1\left(\frac{2\gamma}{\sigma_w^2}|\langle x(t), s(t-\tau)\rangle|\right)}{c(\gamma) I_0\left(\frac{2\gamma}{\sigma_w^2}|\langle x(t), s(t-\tau)\rangle|\right)} \cdot \frac{2\gamma}{\sigma_w^2}\frac{\partial|\langle x(t), s(t-\tau)\rangle|}{\partial \tau}. \qquad (11)
\end{aligned}$$

For the numerical evaluation of the mean square value of (11) and evaluation of estimator $\hat{\varphi}$ and $\hat{\gamma}$, the `\matlab_files\testFiles\eval_CRLB.m` script was used.

Important note on computation of mean square value of (11). For evaluation of $\langle a(t), b'(t)\rangle$ using vectors $\mathbf{a}$ and $\mathbf{b}$ of length `LL` (i.e. their indices are 0,...,LL-1) an equivalent vector $\mathbf{b}'$ is created. It is created as follows.

The result (8) is bounded on vector indices. The bounds come up from the bounds on index of a and b, i.e. $0 \leq n \leq LL-1, \quad 0 \leq n-q \leq LL-1 \quad \equiv \quad n \geq q \geq n-LL+1$:

$$\langle a(t), b'(t)\rangle = \sum_{n=-\infty}^{\infty} a_n \sum_{q=-\infty \neq 0}^{\infty} b_{n-q}^* \frac{(-1)^q}{q},$$

$$= \sum_{n=0}^{LL-1} a_n \sum_{q=n-LL-1\neq 0}^{n} b^*_{n-q} \frac{(-1)^q}{q}. \tag{12}$$

An example for $LL = 7$ is presented. Values of q and $n - q$ are given in the table:

n	0	1	2	3	4	5	6
q	-6	-5	-4	-3	-2	-1	1
	-5	-4	-3	-2	-1	1	2
	-4	-3	-2	-1	1	2	3
	-3	-2	-1	1	2	3	4
	-2	-1	1	2	3	4	5
	-1	1	2	3	4	5	6
n-q	6	6	6	6	6	6	5
	5	5	5	5	5	4	4
	4	4	4	4	3	3	3
	3	3	3	2	2	2	2
	2	2	1	1	1	1	1
	1	0	0	0	0	0	0

Table 2.1. Indices used in computation of the adapted scalar product.

We can notice, that the matrix in the upper part of 2.1 is flipped Toeplitz matrix. The fact is used in the MATLAB implementation:

```
row = 1:LL-1; col = [1 -1:-1:(-LL+1)];
q_idx = flip( toeplitz(row, col), 2);

%index in s-vector, +1 for 1-based indexing
s_idx = (0:LL-1) - q_idx +1;

%equivalent vector to s'(t-tau)
s_tau2_diff_delay = dot(s_tau2_delay(s_idx), (-1).^q_idx./q_idx).';
```

For the computation of mean square value of (11) we will use Weighted Monte Carlo Integration, [5] with noise-samples as the evaluation points. So we will express the two scalar-products using noise vector:

$$\langle x(t), s'(t-\tau)\rangle = \gamma e^{j\varphi} \langle s(t), s'(t-\tau)\rangle + \langle w(t), s'(t-\tau)\rangle \,. \tag{13}$$

$$\langle x(t), s(t-\tau)\rangle = \gamma e^{j\varphi} \langle s(t), s(t-\tau)\rangle + \langle w(t), s(t-\tau)\rangle \,. \tag{14}$$

Monte Carlo Integration evaluates an integral of an integrand function over a region by generating points from the region and computing a (weighted) average of the integrand evaluated in these points. Key-idea is the generating points. The points can be generated from a uniform distribution over the given integration region, if the integrand-values, resp. integrand-shape is not known. But if the integrand-shape is known, at least approximately, it is better to generate points from the region according to a distribution with probability density function (PDF) similar to the integrand-shape as much as possible. The computing of mean value is the case, in which we know approximate shape of integrand – it is the shape of PDF, so we generate points from that distribution. The general form of Monte Carlo Integration is:

$$\int_\Omega g(\mathbf{x})\,\mathrm{d}\mathbf{x} \approx \frac{1}{N} \sum_{i=0}^{N-1} \frac{g(\mathbf{x}_i)}{\mathrm{pdf}(\mathbf{x}_i)}. \tag{15}$$

In our case:

$$
\begin{aligned}
\mathrm{E}[f(X)] = \int_\Omega f(\mathbf{x})\,\mathrm{pdf}_X(\mathbf{x})\,\mathrm{d}\mathbf{x} &\approx \frac{1}{N}\sum_{i=0}^{N-1}\frac{f(\mathbf{x}_i)\,\mathrm{pdf}_X(\mathbf{x}_i)}{\mathrm{pdf}_X(\mathbf{x}_i)}, \\
&= \frac{1}{N}\sum_{i=0}^{N-1} f(\mathbf{x}_i). \qquad (16)
\end{aligned}
$$

We have expressed our examined function (11) by noise samples because the CWGN noise has known PDF and its samples can be easily generated instead of $\mathbf{x}$ (samples of $x(t)$). A code listing from the evaluation of the CRLB for $\hat{\tau}$:

```
N_mean = 10000; %number of noise vectors  for one sigma^2
tauVar = zeros(N_snr,1);
for i=1:N_snr
    sigma2 = gamma^2 /  snr_lin(i); %sigma^2
    w = sqrt(sigma2/2) * (randn(LL, N_mean) + 1i * randn(LL, N_mean));
    sp = h*scProd_s_s + sc_prod_s(w); %scalar product <x(t), s(t-\tau)>

    %scalar product <x(t), s'(t-\tau)>
    sp_diff = h*scProd_s_s_diff + sc_prod_s_diff(w);

    auxCoeff = 2*gamma/sigma2;
    argBess = auxCoeff * abs(sp);
    auxVect = besseli(1,argBess)./besseli(0,argBess) .* ...
        auxCoeff .* abs( sp_diff ) .* cos( angle(sp) - angle(sp_diff) );
    auxVect = auxVect.^2; %E [ ( d^2 ln( p(x|gamma,phi) )/ d phi^2 )^2 ]
    tauVar(i) = 1/mean( auxVect ); %computed variance
end
```

Note, that vectorized operations have been used.

Graphical results follows.

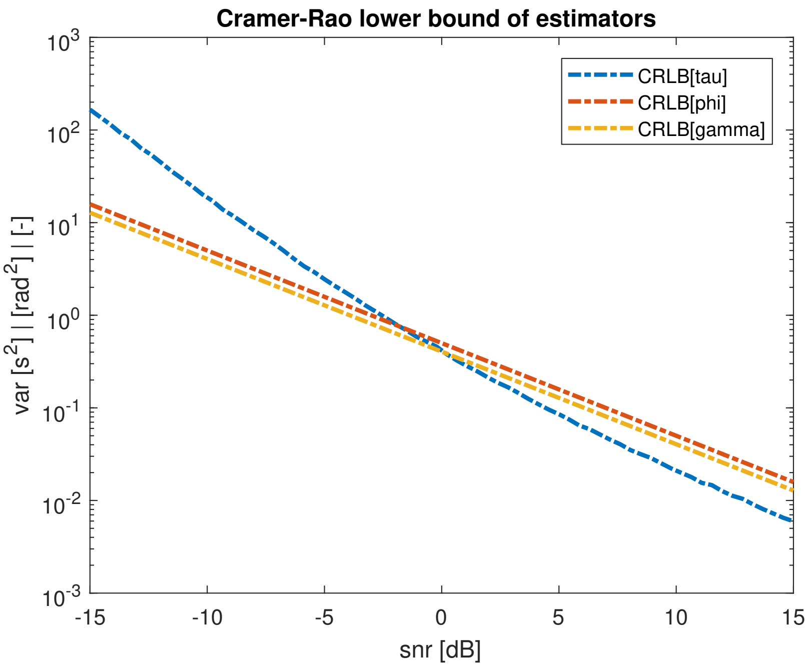

Figure 2.5. CRLB of all 3 estimators: $\hat{\varphi}$, $\hat{\gamma}$ and $\hat{\tau}$.

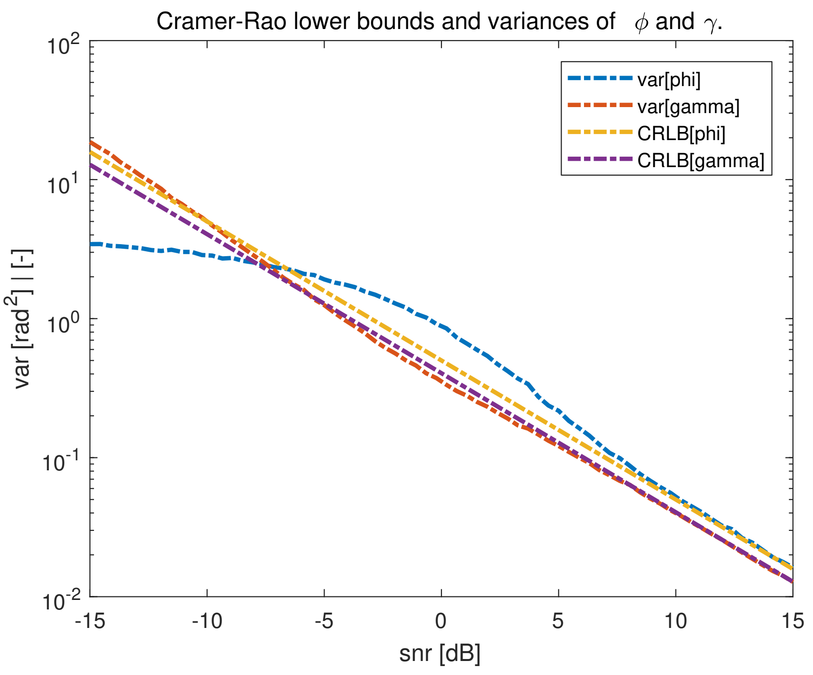

Figure 2.6. CRLB and computed variances of estimators $\hat{\varphi}$ and $\hat{\gamma}$.

In the picture 2.6 we can see the $\hat{\varphi}$ estimator has lower variance than its CRLB for low SNR. It can be caused by phase-folding to the interval $(-\pi, \pi]$. The $\hat{\gamma}$-variance is also lower than its CRLB in vicinity of 0 dB. It can be caused by increased $\hat{\varphi}$ variance – recall, that $\hat{\gamma}$ ML estimator uses $\hat{\varphi}$ estimate. Further, both estimators achieve their CRLB for high SNR.

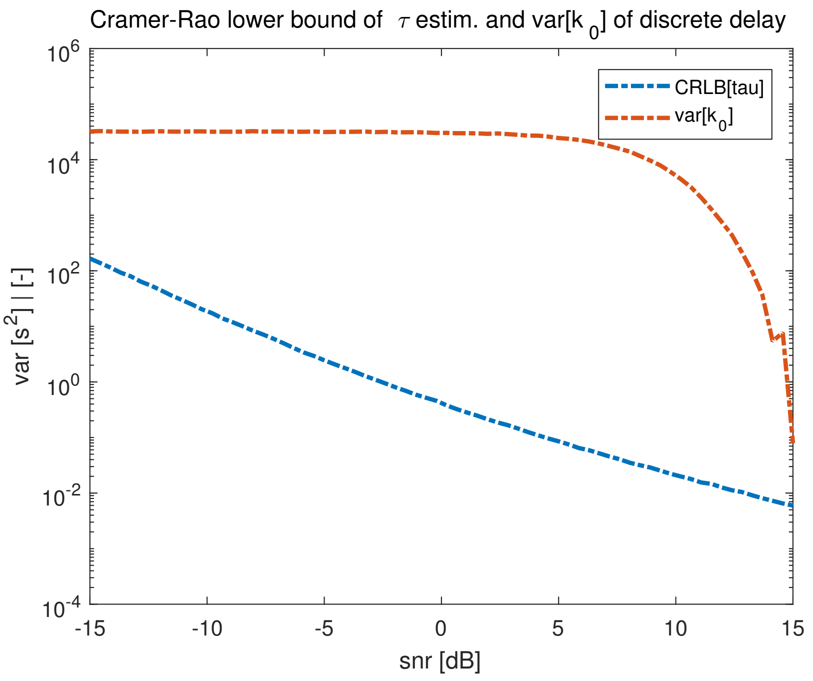

Figure 2.7. CRLB of $\hat{\tau}$ and variance of $\hat{k_0}$ discrete-estimate.

In the picture 2.7 there is depicted the CRLB for $\hat{\tau}$ and variance of $\hat{k_0}$ estimator. We can see no similarity. Clearly, a different approach is needed for performance-evaluation of estimators with discrete parameter space.

The function `matlab_files\func\estim_chanParams.m` has been created for estimation of φ, γ and σ_w^2 from a received signal `x_received` of the channel model $\mathbf{x} = \gamma e^{j\varphi}\mathbf{s} + \mathbf{w}$. Code follows.

```
if any( size(x_received) ~= size(s_orig) )
    %ensure column vector and replicate
    s_orig = repmat( reshape(s_orig,[],1) ,1, size(x_received,2));
end
%first argument is conjugated
scProd = dot(s_orig, x_received);
s_origAbs2 = dot( s_orig, s_orig );
h_phi_est = angle( scProd );
h_abs_est = abs( scProd ) ./ s_origAbs2;
h_est = h_abs_est .* exp(1i*h_phi_est);
sigma2_est = 2* mean( real(x_received - h_est .* s_orig).^2 );
```

Input parameters are

- `x_received` received signal; matrix of size L x N, L – length of vectors, N – number of vectors.
- `s_orig` pilot signal of size L x 1, resp. 1 x L or L x N (if required a different pilot for each vector).

Output parameters are row vectors of $\hat{\varphi}$, $\hat{\gamma}$ and σ_w^2 estimates for each column of `x_received`.

2.5 Frame synchronization

The delay estimator (2) derived and evaluated in previous sections works well if we have whole signal available and we are sure, that it contains synchronization pilot-signal. But in reality we receive signal by parts and we don't know if there is, resp. should be synch. pilot-signal or not. We will design and examine an ad-hoc criterium for deciding whether there is synch. pilot in the received part of signal or not.

For examination of frame-synchronization there was a script
`matlab_files\testFiles\testFrameSynchEnv.m` created. The "-Env" suffix emphasizes synchronization at complex-envelope level, not on symbol level. A short description of the script and examination-results go next. At the beginning a synchronization sequence and a channel estimation sequence are created (pseudorandom BPSK symbols). The pilot samples are created as follows:

- A REC modulation pulse was used with length `Ns_pil` (number of samples per symbol in pilot), i.e. `Ns_pil` samples per (BPSK) symbol will be used. Also there is a possibility to use `2 x Ns_pil` samples per (BPSK) symbol to use narrower bandwidth.
- Synch. sequence is inserted `Nrep`-times (number of repetitions), but starting by the second-half and ending by the first-half of it. Schematically `[ 2 ][ 1 2 ]...[ 1 2 ][ 1 ]`. This scheme – with incomplete sequences at ends – was chosen, because it has better synchronization properties than an ordinary repeated sequence.
- There is an `N_fill` samples between synch. part and channel estimation part left zero.

- The channel estimation sequence is inserted.
- An finally, the REC modulation pulse is applied.

After the pilot samples, there are N_fill zero-samples inserted and next, N_PL_symbs payload BPSK symbols are filled in. The whole packet is inserted to a longer sequence testEnv (of length NtestSeqEnv) starting at index initIdx_pil. At that point, the channel model is applied. The attenuation h is initialized by its phase h_phi and magnitude h_abs. The CWGN noise is of variance sigma2 (σ_w^2). A simulation of receiving testEnv-signal by parts comes next. According to the derived delay-estimator (2) we need to compute dot-product of received signal and shifted version of synch. sequence for every shift. That is achieved by filtering the received signal using a FIR filter. To save a bit of computation-power, we decimate (with no prefiltering) the part of testEnv signal, that is currently under test – testSig. The decimation is done by a factor Ns_pil. The filter impulse-response, consists of reversed sequence of symbols that we are looking for. We are looking for Nrep-1-times repeated synchronization sequence.

```
synchFilt = dsp.FIRFilter('Numerator', ...
      flip(repmat(synchSeq,Nrep-1,1),1).');
```

From the output of filtration of decimated testSig, the squared magnitude is computed. At this time, we have desired vector for maximum-search – corrOutPartAbs2. However, we need a criterium to decide whether the pilot was transmitted. We have created a decision level. The decision level is computed as

```
decLevel(partIdx) = facMean * mean_val + facStd * sqrt(var_val);
```

where mean_val is a mean value of the corrOutPartAbs2 vector and var_val is the variance of the corrOutPartAbs2 vector. The factors facMean and facStd are ad-hoc parameters chosen in initialization section . In the script there is also implemented an exponential decay with parameter fac_oldVar for the decision levels. A comeback to the decision. If all vector-elements are under the level, we decide, no pilot was transmitted and we continue to process next part of testEnv signal. If any element is larger than the level, we decide, there was a transmission and we save the part to a separate matrix framesStored; following parts are stored in the matrix until the N_partsInFrame-number of parts is reached. We use a flag hasReachedBeg to indicate whether a frame beginning was reached. After cycling through whole testEnv signal, we post-process the stored signal in the framesStored matrix.

The post-processing is similar to the "real-time " processing. But unlike the "real-time" processing, it filters the signal from stored matrix framesStored. It selects first few parts (NpartsToCorrel) for the correlation. This time we are looking for the whole envelope not just symbols. The searched envelope-samples are composed by modulation of the Nrep-1-times repeated synch. sequence. The samples are stored in synchSeq_env vector. The filter coefficients consists of reversed synchSeq_env vector-samples. Subsequently after the filtration the magnitude squared is computed and stored in outCorrAbs2_allPilotEnv vector. Next, the outCorrAbs2_allPilotEnv vector is sorted so to find first few maximums. Number of considered maximums is computed as N_peaks*(2*Ns_pil-1), where N_peaks is expecting number of peaks in outCorrAbs2_allPilotEnv. We are interested in the position of the N_peaks peaks. This arrangement was done because in a peak vicinity there are 2*Ns_pil-2 numbers large enough to "shadow" other peaks. Thereafter the absolute maximum (maxIdx_1) is found. We know, the position of other peaks w.r.t. the absolute maximum peak position differs by an integer multiple of `N_synch*Ns_pil`. So we can construct possible position of indices of other peaks. These indices are then intersected by indices of top `N_peaks*(2*Ns_pil-1)` largest values in `outCorrAbs2_allPilotEnv` vector.

```
N_peaks_half = floor(N_peaks/2);
aux_delta = N_peaks_half*N_synch*Ns_pil;
maxIdx_possible_peaks = ...
      (maxIdx_1 - aux_delta): (N_synch*Ns_pil) : (maxIdx_1 + aux_delta);
maxIdx_peaksFinal = intersect(maxIdx_possible_peaks, maxIdx);
```

Finally, the position of the last peak is chosen as a reference. The value is stored in `idxLast_final`. The position is computed w.r.t. the `outCorrAbs2_allPilotEnv` vector, nonetheless we are interested in position-index in original sequence, resp. w.r.t. the beginning of stored parts – `framesStored`. For that we perform a conversion.

```
idxEstAugPL = (idxFirstPart-1)*partLen + idxLast_final ...
- round(N_synch/2)*Ns_pil + (Ns_pil-1);
```

The `idxLast_final` index is converted to the `idxEstAugPL` index (index of estimated augmented payload). By augmented payload we mean channel estimation part of pilot together with payload segment, including `N_fill` initial zeros in front of chan. est. part. The new index is the one after end-sample of the `synchSeq_env` in the whole `testEnv` signal, i.e. the first zero sample of `N_fill` zero-samples between synch. seq. segment and chan. est. segment of pilot.

At the moment, we can extract channel-estimation part and payload part. We demodulate channel-estimation section to symbols and estimate channel parameters. The parameters are estimated both from the whole envelope part and from the demodulated symbols. After running the script, the estimated channel parameters are printed to the command window together with original values. To show, which estimator – either that from demodulated symbols or that from whole envelope – is closer to the original value `o`-sign is put at the more precise estimate.

A note on a computation of correlation. In MATLAB, correlation / convolution can be computed by at least 3 ways:

- By `conv` function.
- By calling `dsp.FIRFilter` object.
- By `filter` function.

The fastest seems to be the `filter` function. The comparison performance of the listed ways to compute correlation / convolution is done in the script: `matlab_files\testFiles\testConvolution.m`.

To perform the synchronization task, the `frameSynchEnv` class was created. It contains algorithms for synchronization described in this section and developed in the `matlab_files\testFiles\testFrameSynchEnv.m` script. The object is created and used at the end of the script. Among other properties one of its properties is a flag `.isFramePrepared` indicating whether the frame is prepared to be returned from the object. The object also implements a method `.getFrames()`, that returns augmented-frame samples. Augmented-frame is a frame consisting of: `N_fill` zero samples, envelope of chan. estimation sequence, next `N_fill` zeros samples and in the end envelope of payload symbols. The object supports also synchronization to multiple sources – `N_SIGS` constructor parameter – but the synchronization is done w.r.t the first stream. The sample-streams are given to the input when called within `step` function. Each steam should be in one column. When synchronized, the method `.getFrames()` returns augmented valid frames for each stream. It also returns `isValidFlags` row vector of logical values. The vector is of length `N_sigs` and contains true for stream with successful acquisition of augmented frame; or contains false for a stream in which the synch. sequence was not found. Synchronization to multiple sources is used in BC stage, when one `frameSynchEnv` object is used for two destinations: D_A and D_B.

An example of output plots will be presented and the content explained.

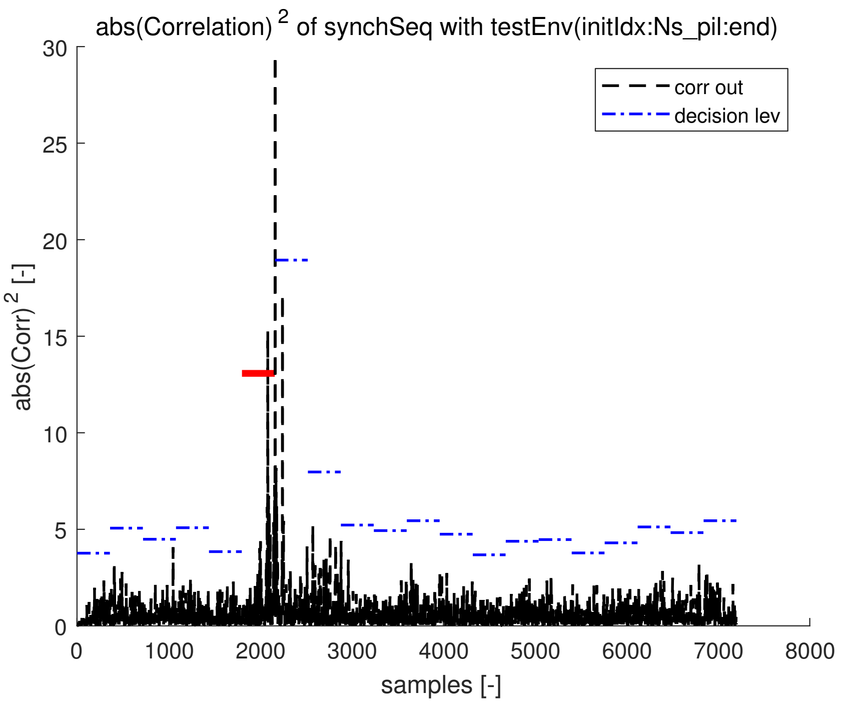

Figure 2.8. "Real-time" synchronization.

In the figure 2.8 we can see the magnitude squared of the filtered `testEnv` signal – the black lines. There was used `Nrep` parameter equal to 3. Blue lines are the decision levels for the corresponding range of samples, width of lines depicts the part range. The red line is a decision level for a part in which the criterium is fulfilled. From that part, the buffering of parts starts. It doesn't matter if the criterium is fulfilled for subsequent parts or not.

In the figure 2.9 we see the post-process synchronization. The correlation is performed with whole synch. seq. envelope and its magnitude squared is depicted in green. Identified peaks are depicted by red asterisks.

2.6 Encoding-decoding

For channel coding there was an LDPC code used. It was chosen from the DVB-S2 standard. In MATLAB there is a function `dvbs2ldpc` that can generate parity-check matrix according to a given nominal rate. However, there is no support for short frames ($n_{\text{ldpc}} = 16200$ bits), so the function `matlab_files\func\dvbs2ldpc_custom.m` has been created as a copy of the former extended by the possibility to generate also short frames. It was necessary just to handle effective rate-values that are not the same as nominal, "q"-values that are not equal to `M/NB` and to copy tables of addresses of parity bit accumulators [3] (`M` is a number of parity bits, `NB` is a number of bits in a block – 360).

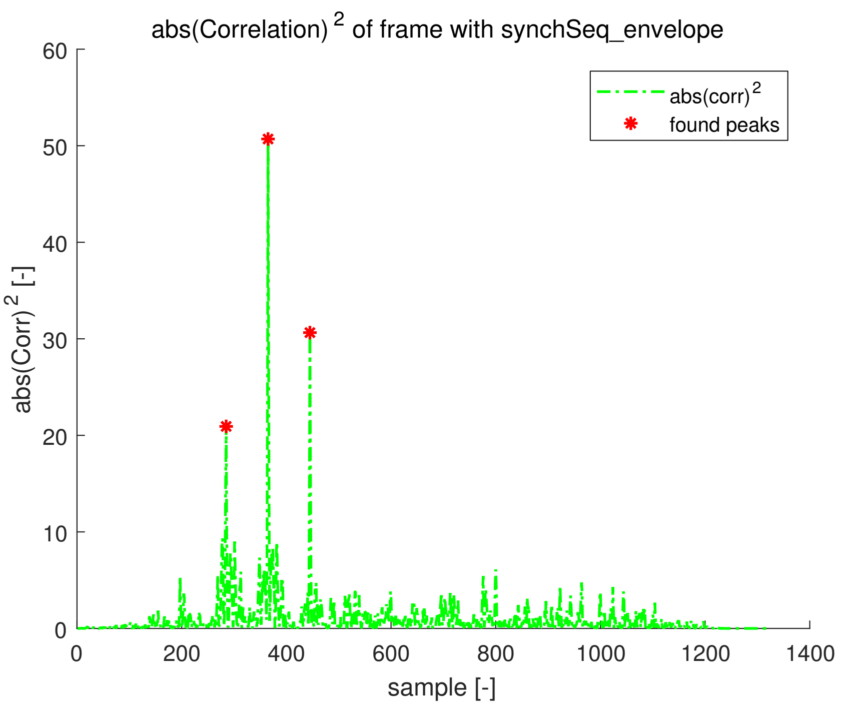

Figure 2.9. “Post-process” synchronization.

For the encoding and decoding MATLAB System objects have been used. For encoding the `comm.LDPCEncoder`, for decoding `comm.LDPCDecoder` object. Both are created using parity-check matrix. When creating decoder there is a possibility to set some additional parameters. Some useful parameters are these:

- a possibility to output whole codeword instead of just an information part (`’OutputValue’, [’Whole codeword’ | ’Information part’ (default)]`).
- maximum number of iterations (`’MaximumIterationCount’, <number> (default 50)`)
- decoding termination condition (`’IterationTerminationCondition’, [’Parity check satisfied’ | ’Maximum iteration count’ (default)]`).

MATLAB System objects are used through `step` function. First input to the function is a handle to a system object. Next inputs are inputs to the system object. For the encoder the input is a data-word (vector of suitable length of logical values); for the decoder the input is Log-Likelihood Ratio (LLR) function computed from received symbols. The LLR is given as: [1]

$$L(c_i) = \log\left(\frac{\Pr(c_i = 0|\text{ channel output for } c_i)}{\Pr(c_i = 1|\text{ channel output for } c_i)}\right). \tag{17}$$

Derivations of LLR for an ordinary one-to-one channel and HMAC channel follow.

For an ordinary one-to-one transmission and CWGN channel model $\mathbf{x} = h\mathbf{s} + \mathbf{w}$ we have symbol-wise channel likelihood function

$$p(x_i|h, s_i) = \alpha \exp\left(-\frac{|x_i - hs_i|^2}{\sigma_w^2}\right). \tag{18}$$

[1] Decoding Algorithm part: `https://ch.mathworks.com/help/comm/ref/comm.ldpcdecoder-system-object.html#bs8gdxn-1`

Because we use one-to-one mapping $s_i = \mathbb{A}(c_i)$ the $p(x_i|h, s_i) = p(x_i|h, c_i)$. For the evaluation of LLR we need a posteriori probability $p(c_i|x_i, h)$, it can be expressed using Bayes rule as

$$p(c_i|x_i, h) = \frac{p(x_i|c_i, h)p(c_i)}{p(x_i)}. \tag{19}$$

c_i symbols are equiprobable, i.e. $\Pr\{c_i = 0\} = \Pr\{c_i = 1\} = 1/2$. So the required ratio is given as:

$$\begin{aligned}
\frac{\Pr\{c_i = 0|x_i, h\}}{\Pr\{c_i = 1|x_i, h\}} &= \frac{p(x_i|c_i = 0, h)\frac{1}{2\Pr\{x_i\}}}{p(x_i|c_i = 1, h)\frac{1}{2\Pr\{x_i\}}}, \\
&= \frac{\alpha \exp\left(-\frac{|x_i - h(-1)|^2}{\sigma_w^2}\right)}{\alpha \exp\left(-\frac{|x_i - h(1)|^2}{\sigma_w^2}\right)}, \\
&= \exp\left(\frac{-(|x|^2 + |h|^2 + 2\Re\{hx^*\}) + |x|^2 + |h|^2 - 2\Re\{hx^*\}}{\sigma_w^2}\right), \\
&= \exp\left(\frac{-4\Re\{hx^*\}}{\sigma_w^2}\right).
\end{aligned} \tag{20}$$

When evaluating LLR, we can use ln instead of log, the result is then scaled by an uninteresting positive constant. Even the number 4 in the formula we could omit, that would cause a decoder performance degradation by a few percent in low SNR. Therefore

$$L(c_i) = -\frac{4\Re\{hx^*\}}{\sigma_w^2}. \tag{21}$$

For an HMAC channel and CWGN model $\mathbf{x} = h_A\mathbf{s}_A + h_B\mathbf{s}_B + \mathbf{w}$ we have symbol-wise channel likelihood function

$$p(x_i|h_A, h_B, s_{Ai}, s_{Bi}) = \alpha \exp\left(-\frac{|x_i - (h_A s_{Ai} + h_B s_{Bi})|^2}{\sigma_w^2}\right). \tag{22}$$

We will use so called Hierarchical Soft-Output Demodulator H-SODEM [2] (Section 4.4), resp. its symbol-wise version. In our case the “soft-output” is the $p(x_i|c_i)$ likelihood function. We use χ_c = XOR and uniform alphabet symbols c_i, that implies [2]:

$$p(x_i|c_i) = \frac{1}{M_c^{K-1}} \sum_{\tilde{c}_i : \chi_c(\tilde{c}_i) = c_i} p(x_i|\tilde{c}_i). \tag{23}$$

(M_c – cardinality of c_i alphabet, K – number of transmitters in HMAC channel) In our case $M_c = 2$, $K = 2$.

$$p(x_i|c_i = 0) = \frac{1}{2}[p(x_i|c_{Ai} = 0, c_{Bi} = 0) + p(x_i|c_{Ai} = 1, c_{Bi} = 1)]. \tag{24}$$

$$p(x_i|c_i = 1) = \frac{1}{2}[p(x_i|c_{Ai} = 0, c_{Bi} = 1) + p(x_i|c_{Ai} = 1, c_{Bi} = 0)]. \tag{25}$$

Similarly as in one-to-one transmission it holds that $p(x_i|c_{Ai}, c_{Bi}) = p(x_i|s_{Ai}, s_{Bi})$ because of bijective mappings $s_{Ai} = \mathcal{A}(c_{Ai})$, $s_{Bi} = \mathcal{A}(c_{Bi})$. The a posteriori probability of the H-symbols is

$$p(c_i|x_i, h_A, h_B, s_{Ai}, s_{Bi}) = \frac{p(x_i|c_i, h_A, h_B, s_{Ai}, s_{Bi})p(c_i)}{p(x_i)}. \tag{26}$$

And the required likelihood ratio for decoding is (conditioning by channel parameters is assumed as implicit)

$$
\begin{aligned}
\frac{p(c_i=0|x_i)}{p(c_i=1|x_i)} &= \frac{p(x_i|c_i=0)}{p(x_i|c_i=1)},\\
&= \frac{\frac{1}{2}[p(x_i|c_{Ai}=0,c_{Bi}=0)+p(x_i|c_{Ai}=1,c_{Bi}=1)]}{\frac{1}{2}[p(x_i|c_{Ai}=0,c_{Bi}=1)+p(x_i|c_{Ai}=1,c_{Bi}=0)]},\\
&= \frac{\alpha\exp\left[-\frac{1}{\sigma_w^2}|x_i-(-h_A-h_B)|^2\right]+\alpha\exp\left[-\frac{1}{\sigma_w^2}|x_i-(h_A+h_B)|^2\right]}{\alpha\exp\left[-\frac{1}{\sigma_w^2}|x_i-(-h_A+h_B)|^2\right]+\alpha\exp\left[-\frac{1}{\sigma_w^2}|x_i-(h_A-h_B)|^2\right]},\\
&= \frac{\mathrm{e}^{-\frac{|x_i|^2}{\sigma_w^2}}\mathrm{e}^{-\frac{|h_A+h_B|^2}{\sigma_w^2}}\left[\exp\left[-\frac{2\Re\{x_i(h_A+h_B)^*\}}{\sigma_w^2}\right]+\exp\left[-\frac{(-2)\Re\{x_i(h_A+h_B)^*\}}{\sigma_w^2}\right]\right]}{\mathrm{e}^{-\frac{|x_i|^2}{\sigma_w^2}}\mathrm{e}^{-\frac{|h_A-h_B|^2}{\sigma_w^2}}\left[\exp\left[-\frac{2\Re\{x_i(h_A-h_B)^*\}}{\sigma_w^2}\right]+\exp\left[-\frac{(-2)\Re\{x_i(h_A-h_B)^*\}}{\sigma_w^2}\right]\right]},\\
&= \exp\left[\frac{-4\Re\{h_Ah_B^*\}}{\sigma_w^2}\right]\frac{\cosh\left[\frac{2\Re\{x_i(h_A+h_B)^*\}}{\sigma_w^2}\right]}{\cosh\left[\frac{2\Re\{x_i(h_A-h_B)^*\}}{\sigma_w^2}\right]}
\end{aligned}
\tag{27}
$$

And finally LLR is

$$
L(c_i)=\frac{-4\Re\{h_Ah_B^*\}}{\sigma_w^2}+\ln\left(\frac{\cosh\left[\frac{2}{\sigma_w^2}\Re\{x_i(h_A+h_B)^*\}\right]}{\cosh\left[\frac{2}{\sigma_w^2}\Re\{x_i(h_A-h_B)^*\}\right]}\right).
\tag{28}
$$

Implementation of both LLR is done in the function `matlab_files\func\llr4ldpcDec.m`. Code of the function follows.

```
if size(h,1) == 1
    llr = -4*real(conj(h) .* x) ./ sigma2;
else
    llr = -4*real(h(1,:) .* conj(h(2,:))) ./ sigma2 ...
        + log( cosh(2* real( conj(sum(h) ) .* x) ./sigma2) ./ ...
               cosh(2* real( conj(diff(h)) .* x) ./sigma2) );
end
```

Its input arguments are:

- Received symbols `x` as a matrix of size `L x N`, N – number of vectors, L – length of vectors.
- Channel attenuation `h` as a matrix of size `1 x N`, resp. `1 x 1` or `2 x N`, resp. `2 x 1`. According to the first dimension it is decided whether to apply one-to-one channel or HMAC channel. If the second dimension is 1, the `h` parameter(s) are applied to every column of `x`.
- Noise power `sigma2` (σ_w^2) as a matrix of size `1 x N`, reps. `1 x 1`. If the second dimension is 1, the same noise power is applied to every column of `x`.

Output is the required LLR of the same dimensionality as input `x`. Performance of the LDPC decoder is done in the script `matlab_files\testFiles\testLDPCDecoder.m`. A poor performance of version for HMAC channel has been noticed for low σ_w^2 ($\sigma_w^2 \approx 5{\cdot}10^{-4}$). It may be caused by a lot of $\pm\infty$ values as a result of low numerical stability of used formula. From the view of the code-word level, decoder receives a corrupted code-word $c_{\mathrm{H,corrupted}} = c_{\mathrm{A}} \oplus c_{\mathrm{B}} \oplus w$. Due to the Isomorphic Layered NCM, the same encoder and decoder can be used to estimate hierarchical code-word and a normal code-word – in one-to-one transmission: $c_{\mathrm{Ncorrupted}} = c_{\mathrm{orig}} \oplus w$.

2.7 Configuration structure

In the simulation there is quite a lot of parameters. To share them easily a configuration structure has been created. The structure is returned from the function `matlab_files\func\nodes_config.m`, resp. `matlab_files\func\nodes_config_short.m`. The `_short` suffix originates from the need to work with frames carrying "short" payload. Instead a frame carrying whole LDPC code-word (in next text, just LDPC frame) a shorter frames were created, each bearing only a part of the whole LDPC code-word. I.e. the whole LDPC code-word was parsed to smaller pieces and transmitted by parts. At the receiver the subframes are assembled. When whole LDPC frame is reached, the decoding can start. The "short" suffix occurs also in names of other files to emphasize they works with short frames. The short version of the config function contains variable `N_PL`, that determines number of payload symbols in one frame. Other variables are identical in both configuration files; some differs only by their values. A configuration structure contains some variables that are generated statically. The static configuration is contained in `.mat` files: `matlab_files\func\nodes_config_static.mat` and its "short" version. The files are created from scripts `matlab_files\func\nodes_config_matFile.m` and also its "short" version. In scripts, vectors for source A, source B and relay node are created. There are vectors such as synchronization sequence, channel estimation sequence, their versions with repeated basic sequence, complex envelopes of the sequences, modulation pulse parameters and samples, and others. Modulation pulse was used RRC – for payload symbols, a function that returns its samples is `matlab_files\func\srrc.m`. The function is taken from [6]. There was made an effort to hold some naming rules:

- `N_` and `N` prefix means a number; number of something.
- `_seq` suffix means the parameter contains a sequence, resp. a vector. Variables with the suffix contain sequence of modulation symbols or a spreading sequence.
- `_env` suffix means the parameter contains an envelope samples.

In scripts, the output of the function `nodes_config` is stored in a variable `cs` (config. structure). The structure is often used as a functions' parameter.

2.8 BitSource, BitEval and bitRelay objects

In any simulation we need a source of information and an evaluator of error. The `bitSource` object was created for the purpose of "creating" data. The object is constructed with one parameter – name of a picture file. The file is being read, every pixel is being quantized according to the half of its range to Boolean values. In the call of `step` function the object accepts one parameter – number of bits to return, and it returns the required bits from the picture. The picture is read column-wise and is periodically extended if needed. In the object there is an index of bit to start reading from, subsequent call to `step` function returns bits starting from the index. When the object is reset, the index is set to 1.

For error evaluation there was the `bitEval` object created. At construction it accepts a picture filename and a flag whether to show received image. It reads the image and expects to receive bits from that picture. At the step-function call, it accepts a vector of logical values. The values are filled-in to an internal array `picData_test` and compared to the corresponding bits of the reference picture (property `picData`). Among other properties there are variables `Nbits` and `Nerrs`. The former contains overall number of processed bits, the latter contains overall number of error bits. When the object is reset, both numbers are set to 0 and index-variable is set to 1.

The script `matlab_files\testFiles\testBitEval.m` was created for demonstration of usage of both objects.

For simulation of a network with relays, it is needed an object that buffers some data and returns it when required. For that purpose, a class `bitRelay2` was created. It is not derived from `matlab.System` class, but rather `handle` class. The number 2 indicates second version. The first version was derived from the `matlab.System` class, but there were problems when input data type changed (from logical to double). The object contains properties from `bitSource` and `bitEval` objects, but does not enable cyclic overwrite of the contained picture. The performance of `bitRelay2` is demonstrated in the script `matlab_files\testFiles\testBitRelay.m`.

For multiple sources and sinks there were created versions of `bitSource` and `bitEval` with `_multi` suffix. Their usage is demonstrated in `matlab_files\testFiles \test-BitEval_multi.m` script.

2.9 TxEmul and RxEmul objects

As mentioned in the beginning of the chapter, we are designing 2 types of simulation: 1) a pure computer simulation and 2) an over-the-air simulation. In this section we will focus on the former one, specifically on the transmitter and receiver emulation.

The pure computer simulation was developed as two separate processes running MATLAB. The communication between the processes was done by means of a shared file. To imitate the API of the software radios, two objects have been developed. The `TxEmul` object is for transmission and `RxEmul` object is for reception using a shared-memory file. Creation parameters of both objects are a name of a shared-memory file, a number of complex numbers to be able to store in the file and a `WRITTEN_FLAG` – a number used as a sign, that a process using this flag has written something to the shared file. TxEmul and RxEmul objects of one process should have the same flag – say it is a number WF_1. TxEmul and RxEmul objects of another process should use a flag, that is the opposite to the other-process flag, i.e. $\mathrm{WF}_2 = -\mathrm{WF}_1$. Rx object has an additional creation-parameters – `FRAME_SIZE` – a number of complex samples to return when called within the `step` function and a number of output channels `NUM_OUT`. Output streams are in columns of an output matrix. A "fraud" has been committed, in the sense that the output signal for all channels is the same. A matrix of `NUM_OUT` columns and `FRAME_SIZE` rows is returned. Using parameter `numOut` bigger than 1 is intended as a Broadcast Channel stage.

When the objects are being created, first it is checked if the file exists. If it exists, an error is reported when it doesn't have enough space for storage – if so, you should set the handle to the file (`.mmFile` property) to `[]` or an easier option: just clear objects using that file and finally remove the file. Then the creation should work. If the file doesn't exist, it is created and mapped. The file is accessed as a matrix. First few numbers in the file are used as a header. The header contains information about

- A number of complex numbers currently written and not read yet.
- An offset in the file for reading by parts.
- A written flag – indication of a state. The flag is used as a signal, resp. mutex.

When calling the `TxEmul` object with `step` function, the object accepts 3 parameters:

- A matrix of input data – `inSamples`. One column for each source.

- Channel attenuation parameters – h_params. It should be a row vector – for each source one number.
- Noise samples – w. It should be a column vector of same length as data.

TxEmul object simulates MAC channel by receiving inSamples matrix that has 2 or more columns. Within step function call it accepts also channel attenuation parameters h_params and noise samples w. h_params should be a row vector of the same length as the number of input channels – number of columns in the inSample matrix. It applies the channel model:

```
sum( repmat(h_params,size(inSamples,1),1) .* inSamples, 2) + w; %AWGN
```

and saves the the samples to the shared file. MATLAB does not enable to store complex numbers, so the real and imaginary parts are separated, a new matrix is composed and stored to the file in its linearized form:

```
outSigMat = [real(outSig).';imag(outSig).'];
obj.mmFile.Data(obj.dataIdx : obj.dataIdx+2*dataLen-1) = outSigMat(:);
```

Then a setting of the header of the shared file follows.

The RxEmul object takes no input parameter.

Usage of both objects is demonstrated in scripts matlab_files\testFiles \testTxEmul.m and matlab_files\testFiles\testRxEmul.m. These scripts should be executed from separate processes running MATLAB.

2.10 Tx and Rx pure computer simulation, starting point

In this section, there will be an introduction to the pure computer simulation given. There will be presented m-files for simulation of Tx and Rx. The m-file matlab_files\testPair\testChainTx.m contains transmitter simulation and matlab_files\testPair\testChainRx.m contains receiver simulation. It is necessary to use at least two separate processes running MATLAB. One script should run in one process, the second script in another process. Be sure to run the scripts from the matlab_files\testPair folder. Run scripts by sections – first the initialization section of both, then %% send data in Tx and %% receive data in Rx.

A picture is transmitted and evaluated. The picture is sent by a "long" frames, i.e. whole LDPC frame at once. In each loop cycle there are printouts of current iteration number and channel parameters used/estimated. A description of a transmitter iteration loop follows:

1. N_InfoBits is taken from the picture and stored in bitFrame.
2. The bitFrame is scrambled by a scramble sequence cs.scrambleSeq and encoded by an LDPC encoder. The ldpcFrame of length cs.N_ldpc is returned.
3. A mapping of payload data to phyFramePL – constellation symbols is executed.
4. phyFrame_env – complex envelope of the phyFramePL symbols is constructed using RRC modulation pulse and cs.pulse_Ns samples per symbol. For modulation of symbols a function modulate_symbs was created – it supports modulation of multiple symbol-frames.
5. wholeFrame_env vector is composed of a pilot envelope, a guard zeros samples and phyFrame_env samples.

6. Some initial and final zero-samples are added to the `wholeFrame_env` samples, to stimulate frame synchronization.
7. A noise is generated and the frame is "transmitted" using `TxEmul` object.

Now, we will describe a receiver iteration loop:

1. Frame synchronization object is reset – flags and filter states are set.
2. A while loop iterates as long as a frame is not prepared. It calls Rx object and frame synch. object.
3. The `phyFrameAug` augmented frame is acquired from frame synch. object.
4. Next, channel estimation part is extracted to the `xChanEst` variable. Position indices of channel estimation part in augmented frame are stored in config. structure in a substructure `.idx`. Names of fields of the substructure follow the rule: `[H|N]_[PL|ChE]_[i|f]` – first letter indicates hierarchical frame or normal frame, second part indicates payload or channel estimation part and the third part indicates initial or final index.
5. `xChanEst_symbs` channel estimation symbols are acquired by demodulation of `xChanEst` signal. For demodulation of symbols from complex envelope the function `demodulate_symbs` was created, as the function `modulate_symbs` it also supports multiple envelope-frames.
6. Channel parameters are estimated.
7. Payload envelope part is taken from the augmented frame.
8. Payload symbols are acquired from the payload envelope.
9. Log Likelihood Ratio for LDPC decoder is computed.
10. A code word estimate is produced.
11. Data word estimate is computed as a xor of an estimated code word and a scramble sequence.
12. Finally, data word is evaluated in bitEval object.

Chapter 3
Software radios

In this chapter we will describe software radios used for simulation. We will take a look at their interface with computer, usage in MATLAB, their parameters and others.

Software radios (SR), used for experiments, are devices connected to a computer through an ethernet cable through a network switch. Every radio has its own antenna and some additional inputs e.g. for timing and frequency synchronization. Through the ethernet cables, there are complex envelope (CE) samples transmitted. SR Tx modulates the CE to a desired center frequency and transmits it through its antenna. SR Rx demodulates received signal from a center frequency to a CE, samples it and sends samples to the computer. So computer communicates with radios by means of CE samples.

Our software radios are of type USRP N210 [1] from Ettus Research company. REF CLOCK input is for frequency synchronization and PPS IN (pulse per second signal) input is for timing synchronization. For the USRP N210 model the REF CLOCK is fixed at 100 MHz. PPS input is optional, it helps to synchronize two or more radio devices at simultaneous transmission. We have also a clock generator for synchronization of the radios. All radios are connected to the computer through a 1Gb ethernet switch. In the picture 3.1 we can see software radios used for the experiments.

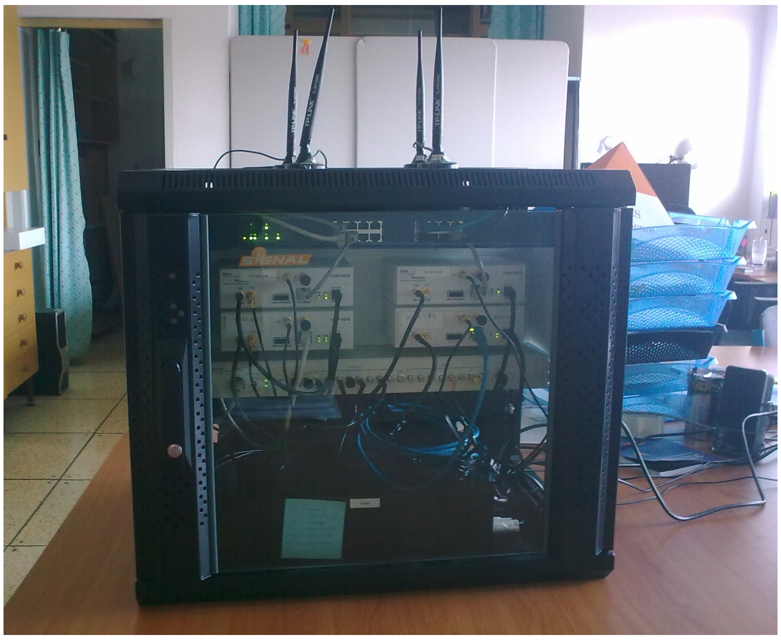
Figure 3.1. A photo of software radios used for the experiment.

[1] http://www.ettus.com/wp-content/uploads/2019/01/07495_Ettus_N200-210_DS_Flyer_HR_1.pdf

3.1 MATLAB interface with software radios

MATLAB provides an add-on of Communications Toolbox for support of USRP radios [1]. It can be downloaded directly from the site or through the MATLAB environment: Home → Add-Ons → Get Add-Ons. Look for "USRP Radio". The download is often not successful, a certain workaround is needed. The add-on contains among other things the transmitter `comm.SDRuTransmitter` [2] and the receiver `comm.SDRuReceiver` [3] system objects. These objects are used for communication with radios in MATLAB. Next we will describe some properties of these two objects.

SDRuTransmitter object properties:

- `Platform` – For USRP N210 the property should be `'N200/N210/USRP2'`.
- `IPAddress` – `<ip_address_string>`. IP address(es) of the radio. When there is a requirement for simultaneous transmission of more radios at once, the property contains more IP strings separated by commas, e.g. `'192.168.10.101,192.168.10.105'`.
- `ChannelMapping` – For USRP N210, it is a row vector 1 up to number of radios used for simultaneous transmission (NumR). When simultaneous transmission is selected. The SDRuTransmitter at `step` function call, expects a matrix of NumR columns. It uses the mapping from the property to assign columns to the radios.
- `CenterFrequency` – Desired center frequency for transmission in Hz. Default value is 2.45 GHz.
- `PPSSource` – Source of PPS signal `['External'|'Internal']`.
- `ClockSource` – Source of a 100 MHz clock signal `['External'|'Internal']`.
- `InterpolationFactor` – Factor of interpolation. The radios processes the samples at 100 MHz, but it is possible to send the data to radio at lower rates. The radio interpolates the samples by `InterpolationFactor` and then converts to analog, modulates and transmits. The maximal interpolation factor possible is 512. So the minimal data rate of samples is $100 \cdot 10^6/512 \approx 195\,\text{kHz}$.
- `Gain` – overall transmission gain involving discrete and analog part.
- `TransportDataType` – `['int8'|'int16']`. Transmitter object expects samples of magnitude lower than 1. Every sample in input vector is quantized to the `TransportType`. E.g. when `'int8'` is selected, for real part there are 8 bits and for the imaginary part there are also 8 bits, for each sample.
- `EnableBurstMode` – `true | false`. Burst mode of transmission is a way of sending data such, that there are frames buffered until a limit and transmitted as a whole at once.
- `NumFramesInBurst` – `<number> (default 100)` A number of frames in one burst. When selected `EnableBurstMode` to true, it buffers `NumFramesInBurst` until real transmission. So a real transmission is done in every `NumFramesInBurst`-th call of the SDRuTransmitter within the `step` function.

SDRuTransmitter object at step function receives vector or matrix of samples. In our case – of BPSK modulation – pure real samples, but it can receive also complex samples. When called for the first time within `step` function, the object is initialized to a type of the input and you cannot change it until you release the object. So e.g.

[1] `https://ch.mathworks.com/matlabcentral/fileexchange/40406-communications-toolbox-support-package-for-usrp-radio?s_tid=prof_contriblnk`

[2] `https://ch.mathworks.com/help/supportpkg/usrpradio/ug/comm.sdrutransmitter-system-object.html`

[3] `https://ch.mathworks.com/help/supportpkg/usrpradio/ug/comm.sdrureceiver-system-object.html`

when you first transmit pure real data and then you want to transmit complex data, it complains about errors of data-type change.

From trial tests performed, it has been found that even though the property `EnableBurstMode` is set to false, there is a certain minimal number of samples to be buffered before real transmission. E.g. for `InterpolationFactor` 512 it was about 8000 samples buffered before real transmission (buffer settings were unknown).

SDRuReceiver objects has similar properties as SDRuTransmitter. Their enumeration follows:

- `Platform` – For USRP N210 – `'N200/N210/USRP2'` value.
- `IPAddress` – `<ip_address_string>` IP address(es) of controlled radio(s). Same as Tx. If multiple simultaneous reception is required, the IP addresses of used radios are separated by commas.
- `ChannelMapping` – vector – a permutation of numbers $1..N_{\mathrm{Rx}}$, where N_{Rx} is the number of receivers controlled by the object. When the object is called within `step` function it returns a matrix of N_{Rx} columns. The mapping controls assignment of IP address to the column of output matrix.
- `CenterFrequency` – `<number> (default 2.45e9)`. Center frequency in Hz for reception.
- `PPSSource` – `['Internal'|'External']` Source of pulse per second signal.
- `ClockSource` – `['Internal'|'External']` Source of 100 MHz clock signal.
- `SamplesPerFrame` – `<number> (default 362)`. Length of returned vector when the object is called within `step` function (number of rows in returned matrix).
- `Gain` – `<number>` Overall gain of discrete and analog part.
- `DecimationFactor` – `<number>`. The receiving radio generates samples at 100 MHz, decimates them by this factor and sends to the computer. Maximal decimation factor is 512.
- `TransportDataType` – `['int16'|'int8']` Data type used for transport from radio to PC. Radio quantizes real and imaginary part of input samples (of magnitude lower than 1) to `int16` or `int8` data type.
- `OutputDataType` – `['double'|'single'|'Same as transport data type' (default)]`. Data type of returned matrix. The SDRuReceiver object performs conversion from the `TransportDataType` to `double` or `single` precision real numbers (real and imaginary part). Or it lets the type same as `TransportDataType`.
- `EnableBurstMode` – `[true | false]`. Similar property as in Tx object. When true, it buffers `NumFramesInBurst` frames and returns first frame. Subsequent call of `step` function returns the next buffered frame until all are read. When the buffer is empty and the `step` function is called, it again buffers `NumFramesInBurst` new frames.
- `NumFramesInBurst` – `<number>` Number of buffered frames per one real samples-acquisition process.

The SDRuReceiver returns also a length (number of rows) of output vector / matrix of samples. The length is either 0 or `SamplesPerFrame`. The 0 indicates unsuccessful reception. The event occurs at first few initial calls of the object. Because of it there is used a construction:

```
len = 0;
while( len <= 0 )
    [phyEnv, len] = step(rxObj); %envelope of phy frame
end
```

instead of just a direct call.

Some useful application notes [1] follows. When using SDRuTransmitter and SDRuReceiver objects, some warnings can appear. One of the most often are `UHD: U` and `UHD: O`, resp. `UHD: D` warning messages. `UHD: U` warning message indicates the Underflow, i.e. SDRuTransmitter did not produced data at a rate required by radio. `UHD: D` warning message indicates the Overflow, i.e. SDRuReceiver object did not processed received data fast enough. On Linux systems it can help to start MATLAB as a privileged user. That can increase thread priorities.

There are several examples for Simulink and pure MATLAB [2] that demonstrates usage of these objects. There is also possible to assign one physical device to SDRuTransmitter and at the same time to the SDRuReceiver, but only from one process.

3.2 Radio performance

For initial transmission and reception tests, scripts `matlab_files \testPair \testChainTx_HW.m` and `matlab_files\testPair\testChainRx_HW.m` were created. They are derived from the versions without `_HW` suffix. `TxEmul` and `RxEmul` objects are replaced by `comm.SDRuTransmitter` and `comm.SDRuReceiver` objects. The scripts are not written in a nice way, there is a lot of commented-out code, but these scripts have been used for an acquisition of useful measurements.

First measurement was aimed at frame synchronization. In one process a transmitter was transmitting in a loop and in second process a receiver with frame synchronizer were launched in a loop. At the beginning default number of `SamplesPerFrame` was chosen (362). After several attempts the synchronization was successful, but the received frame was composed of several subframes, not continuing its predecessor subframe. In the figure 3.2 we can see the original frame samples that were sent (blue line) and received frame samples (red lines). The received frame is cut and aligned to the original frame so that the cut-parts match the original frame samples. For the visualization of received (complex) samples only a real part is displayed. A scaling by a constant is also applied so to match the amplitude of the sent signal.

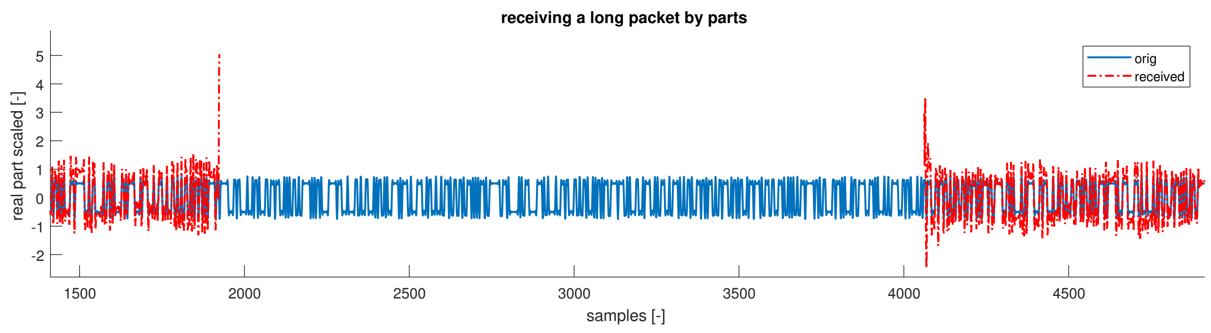

Figure 3.2. Measurement 1, unsuccessful reception of a whole frame by parts.

We can notice a peak in the place of the cut and a little oscillations of amplitude of received signal. The oscillation of amplitude is caused by a little misalignment of center frequencies of receiver w.r.t. the transmitter. The cause of origin of the peak is not known to me.

In the figure 3.3 we can see the same frames as in the figure 3.2. Their initial parts are visualized. We can see the separator between pilot-synch. part and pilot-channel

[1] `https://files.ettus.com/manual/page_general.html`
[2] `https://ch.mathworks.com/help/supportpkg/usrpradio/application-specific-examples.html`

estimation part – it is a `N_fill` = 5 zero samples near the 480-th sample. Further, we can notice pilot deformation. The deformation could be particularly caused by an insufficient bandwidth of transmitter and receiver. The thinnest rectangle pulses are of width 2 samples. We see, parts of the thinnest pulses are almost completely erased but only in one polarity, that is remarkable. I think it may be caused by an "effort" of system to preserve zero mean value of signal samples. We can see that majority of wide pulses have positive amplitude. The effect of non null mean value is enhanced by repetition of base synchronization sequence in pilot.

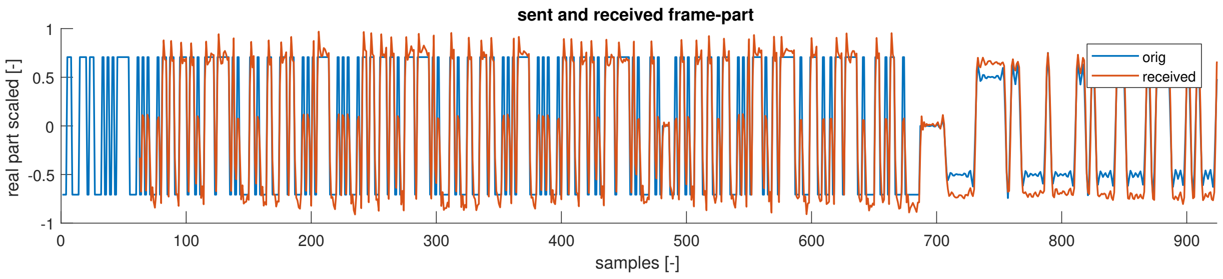

Figure 3.3. Measurement 1, pilot part is displayed.

In this measurement, there were parameters set as follows:

- `pilot_Ns` = 2 samples per symbol in pilot envelope.
- `InterpolationFactor` and `DecimationFactor` were set to 50.
- `Gain` of transmitter and receiver were maximal possible. Tx gain was 31.5 dB and Rx gain was 38 dB.

With this settings the second measurement was performed.

The second measurement was targeted to the pilot deformation. With the previous setting several frames have been acquired and pilots have been extracted. In the figure 3.4 we can see 4 pilots and the original signal. Again only scaled real part of received signals is visualized.

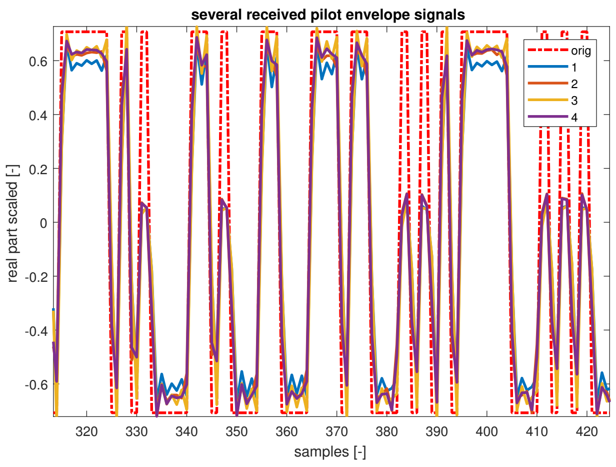

Figure 3.4. Measurement 2, pilot deformation.

We see the same phenomenon as in the figure 3.3. Non-null mean of the original signal caused that thin peaks are almost erased in one polarity.

The same measurement was done also for settings:

- Effectively 4 samples per symbol in pilot envelope.
- `InterpolationFactor` and `DecimationFactor` were set to 512.
- `Gain` of transmitter and receiver were maximal possible. Tx gain was 31.5 dB and Rx gain was 38 dB.

The result of measurement is in the figure 3.5. And again there is an effect of non-null mean value of original pilot signal.

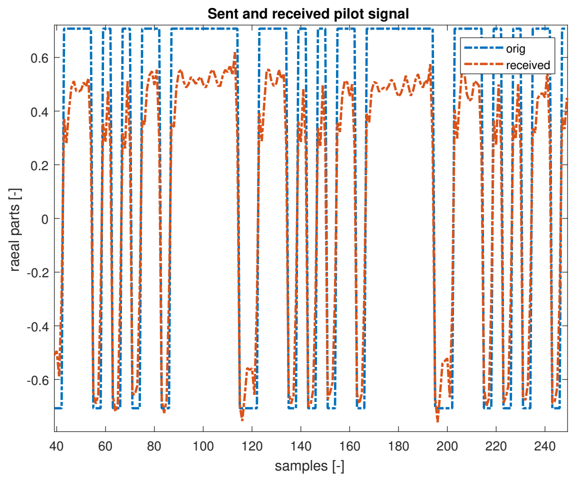

Figure 3.5. Measurement 2, pilot deformation with another pilot signal.

After the performed tests. The function `matlab_files\func\getRandZeroMean.m` was created. It generates random BPSK symbols with 0 mean value. It generates `N_rows/2` random indices. The indices are used to set a value 1 in the output matrix. The matrix is of the size `N_rows x N_cols`. Other indices are set to -1. For indices generation a built-in function `randperm` was used. The function `getRandZeroMean` is applied in generation of static configuration, i.e. in files `nodes_config_matFile[_short].m`.

After the generation of new zero-mean pilot signals. The test was run again. Results for the 2 samples per symbol in pilot envelope are in the figure 3.6, results for 4 samples per symbol in pilot envelope are in the figure 3.7. For both measurements, the interpolation and decimation factor was 512.

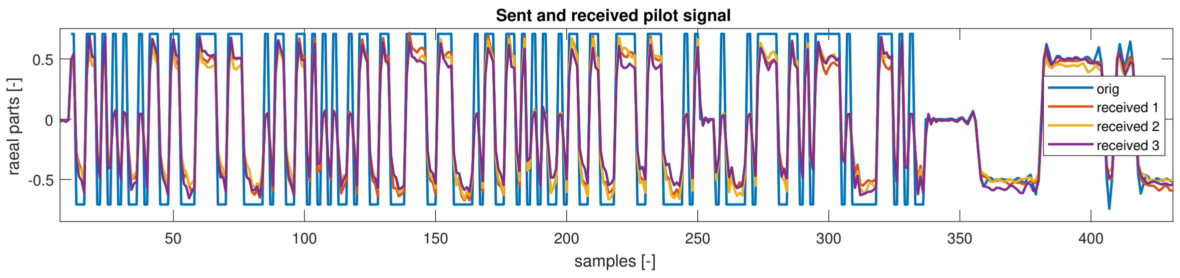

Figure 3.6. Measurement 3, pilot deformation with a zero-mean pilot signal using 2 samples per symbol in pilot envelope.

Conclusion from the measurements is that there was made a mistake when it was not checked the mean value of the pilot signal. In the code, there was blindly used `randi` random number generation function to generate synch. sequence and channel

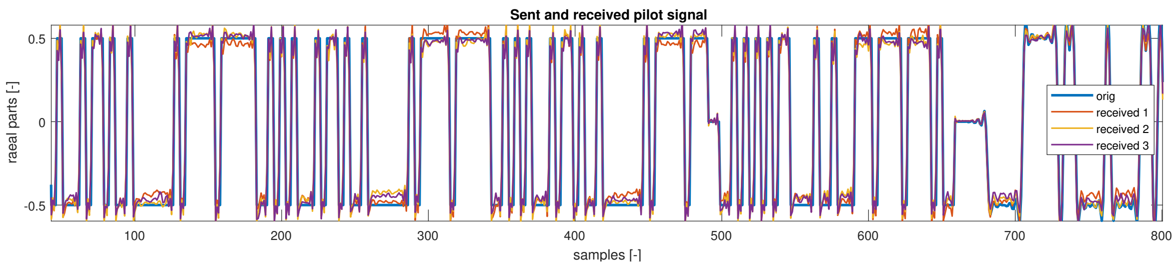

Figure 3.7. Measurement 3, pilot deformation with a zero-mean pilot signal using 4 samples per symbol in pilot envelope.

estimation sequence. The mean value was emphasized by repetition of synchronization sequence in the pilot. The overall system bandwidth seems to be enough for 4 samples per symbol for rectangular pulse used. Finally, it is better to receive longer frames – at least 3-times longer than length of the transmitted frame – than try to receive and process successive short frames (shorter than the transmitted one). Even though, the acquiring and processing of consecutive short frames can be achieved most of the time and on Linux systems in the root regime even more often, there is no warranty.

Chapter 4
Simulation performance and results

In this chapter we will describe final simulation files for the pure computer simulation and the simulation with software radios. We will discuss simulation results and problems that occurred. There was one more level of simulation designed before butterfly network simulation – a base-level simulation. Simulation files of the base-level simulation should serve as a skeleton for simulation of more complex networks.

4.1 Base-level simulation

The concept of base-level simulation uses one pair of transceiver and receiver in both processes (simulation files). The goal is to exchange data between these processes. Each process should transmit its image a receive an image of the second process. In the pure computer simulation, processes are synchronized using a flag in the shared file, but in the over-the-air simulation there is no such flag. Because of that, there was a modified communication protocol ARQ Stop & Wait implemented [7]. In the original version of ARQ Stop & Wait protocol there is one sender on one side and one receiver on the other side. Sender sends its message and waits until it receives an acknowledgement (ACK) from receiver. If the sender does not receive an acknowledgement until a specific time, it resends its message. The receiver also waits for message until a specific time and if it does not get it, it resends ACK to sender. The receiver sends acknowledgements only. There are two versions of ACK to indicate sequence number of sent frame. Without the two versions of ACK, it can happen e.g. that sender receives ACK for last but one frame, but it thinks it is an ACK for last frame (the Rx resent ACK for last but one frame, because it has not received the last one), so sends a new frame. Hence, the receiver will miss one frame. The protocol is modified in a sense that unlike in the original case the communication is peer2peer. Together with a message, there is an ACK sent also.

In our case the ACK's role is fulfilled by 2 versions of channel estimation sequence. In the configuration files `matlab_files\func\nodes_config_matFile[_short].m` there is `chanEst_seq1` vector besides the `chanEst_seq`. Implicitly there is `chanEst_seq` used in pilots and if required it is changed for `chanEst_seq1`. The ACK is handled by two functions

- `matlab_files\func\addAckMark.m` changes implicit `chanEst_seq` in pilot envelope signals for `chanEst_seq1` in every second loop iteration.
- `matlab_files\func\isFrameAcknoledged.m` checks whether it was received a right ACK. First it checks which channel estimation sequence was used. The task is similar to frame synchronization task. We are looking for a sequence in the received signal. In this case we don't alternate delay of vectors, but rather versions of looking-for sequence. The more probable is the one channel estimation sequence with bigger absolute value of its scalar product with received symbols.

The functions are designed in a way to handle signals from all kinds of network stages (MAC, BC and also ordinary one2one stage).

Next, there will be simulation scripts presented. The base-level simulation is done in files `matlab_files\testPair\tCh_short[1|2]_ack_v2.m`. The main loop consists of 4 basic parts / operations, they are:

1. data preparation
2. transmission
3. reception
4. processing of received data

The second script contains the same operations in its main loop – also in the same order – but it starts the main loop with reception and after the main loop ends, it sends last message. For data preparation and processing of received data, there were separate functions created. These functions are

- `matlab_files\func\getBitsToSend_short.m` for returning bits that will be directly modulated and sent, and
- `matlab_files\func\processFrame_short.m` for collecting and processing of received samples.

Both functions are used to handle correct decomposition and composition of a whole LDPC frame from subframes ("short" frames). In these functions, there are *persistent* [1] variables used (equivalent of `static` variables in C language). This feature enables us to use benefits of global variables (such as storing current state of a variable) without their negatives (their open access from workspace). Again, as functions for ACK management, also these functions are written in a general way to handle frames in all possible stages (HMAC, BC and one2one). Next, a description of both functions follows.

Function `getBitsToSend_short.m`. It stores current data-word and code-word bits, together with an index of lastly access code-word bit. Decides according to the index value whether there is a need to create a new code-word or there is a demand to access next bits in the current code-word. And returns required number of bits from each code-word (in a matrix of dimension `N_bits` x number of sources).

Function `processFrame_short.m` processes augmented frames. Recall that augmented frame is the PL envelope augmented by channel estimation envelope and `N_fill` initial zeros – as returned from frame synch. object. It also distinguish between normal frame and hierarchical frame. The decision is made according to the length of input frame(s). First, there is extracted channel estimation part of envelope using precomputed indices. These indices are stored in configuration structure in a substructure `.idx`. The `k0_est` variable relates to the H-frame. It is an estimate of delay of the second source w.r.t the first source. This will be explained later in this chapter. Further, if the frame is H-frame, the channel estimation part of the second frame is extracted and concatenated to the channel estimation part of the first source. Channel estim. parts are held in `x_chanEst_env` matrix. Next, channel estimation symbols are demodulated and there is made a decision which channel estimation sequence was used, it should be same for all sources / columns. Using this sequence, channel parameters are estimated. Payload extraction and demodulation follows. From the demodulated symbols, LLR is computed, filled to the persistent variable `llr_whole` and if this was last subframe a decoding follows. After decoding, there is xor with scramble sequence applied if the frame is not H-frame. In H-frame, decoded H-data-word does not have to be scrambles, because the scramble sequence cancels out: $(b_\mathrm{A} \oplus b_\mathrm{scr}) \oplus (b_\mathrm{B} \oplus b_\mathrm{scr}) = b_\mathrm{A} \oplus b_\mathrm{B}$, as

[1] `https://ch.mathworks.com/help/matlab/ref/persistent.html`

opposed to the one2one decoded bits when there is a need to apply scramble sequence to data-word so to extract unscrambled version of data-word: $(b_A \oplus b_{scr}) \oplus b_{scr} = b_A$. Unscrambled data are then evaluated.

Next we will mention the `matlab_files\func\prepare_envSamp_short.m` function. The function constructs envelope(s) samples of a normal frame or H-frame based on the dimension of input matrices. It accepts payload bits to modulate, pilot envelopes samples and configuration structure. Every column of `data_bits` and `pil_samps` corresponds to a new source. If there are multiple sources, pilots are placed so that they are orthogonal in time and there is `N_fill` additional zero samples inserted between each subsequent pilots. Payload bits are mapped to BPSK symbols ($0 \to -1, \quad 1 \to 1$) and modulated using RRC pulse and `pulse_Ns` samples per symbol. Finally, whole frames are composed by concatenation of pilots and payload envelopes.

In the scripts, there are also some simple functions used: `matlab_files\func \gen-Noise.m` generates samples of CWGN of the given variance and according to a required size or output matrix. Function `matlab_files\func\prepAppZeros.m` prepends and appends zeros to the input matrix. This is used to employ the frame synchronizer. And function for initialization of objects: `matlab_files\func\initObjs.m` – it initializes objects according to its class / type and supports variable number of inputs.

Finally, there is also a function `matlab_files\func\waitForFrame.m` used. The function accepts a receiver `rxObj` handle, a frame-Synchronizer `frSynchObj` handle and a number of trials to perform `N_iter`. It tries to synchronize in the first `N_iter` frames – returned by the receiver object. If the frame beginning was reached within these `N_iter` trials, it continues until a whole frame is prepared in frame synchronizer object. This function works with `RxEmul` and `comm.SDRuReceiver` receiver object. In the end of the function there is a test whether a frames have been found; if yes, (augmented) frames are returned together with their validity flags.

In the beginning of both scripts there are 7 objects created: `bitSrcObj`, `bitEvalObj`, `ldpc_enc`, `ldpc_dec`, `txObj`, `rxObj` and `frSynchEnv`. Names of used variables are the same in both scripts. Be sure to run the script from the `matlab_files\testPair` folder. First, run the initial section of both scripts then `%% one2one transmission` sections. In the figure 4.1 we can see example outputs of the mentioned scripts. The simulation settings was as follows. Script 1: channel parameters from 1 to 2: $h_A = 0.92 \cdot e^{j1.9}$, $\sigma_w^2 = 0.89$; Script 2: channel parameters from 2 to 1: $h_A = 0.96 \cdot e^{j1.3}$, $\sigma_w^2 = 0.91$.

Left picture of the figure 4.1 was received in script 1 with 6% of error bits. Right picture of the figure 4.1 was received in script 2 with 7% of error bits.

For the over-the-air simulation, there were derived scripts used. The scripts have a suffix `HW`: `matlab_files\testPair\tCh_short[1|2]_ack_v2HW.m`. In these scripts, only receiver and transmitter objects are substituted by `SDRuReceiver`, resp. `SDRu-Transmitter`. In the scripts, Tx and Rx objects use the same device (same IP address is assigned to Tx and Rx in one script), but there were also tests where the devices are different.

The simulation did not work entirely as expected. In the figure 4.2 we see the most successfully received images. When the transmission of the whole LDPC frame was performed by parts, insertion of an error subframe happened often, that broken decoding of the whole frame. So there was transmission of the whole LDPC frame selected.

Substantial settings of the over-the-air simulation, from which the pictures 4.2 result:

- `N_PL` = `N_ldpc`, i.e. whole frame was transmitted at one call of Tx object.

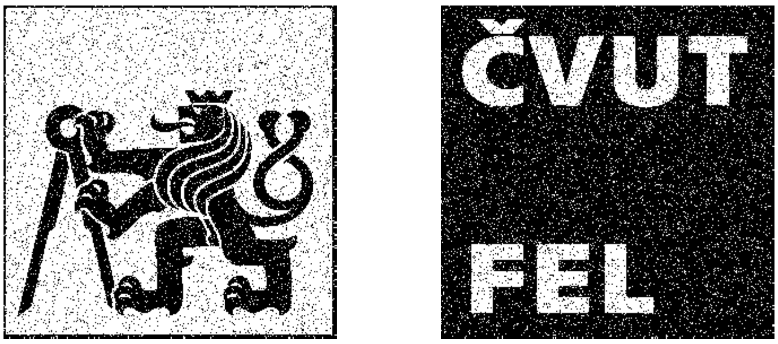

Figure 4.1. Example of received pictures as an output from base-level simulation.

- `rxFrameSize` = 375000 samples – maximal length of received frame (of Rx object) have been chosen (Rx parameter: `SamplesPerFrame`).
- `N_iter` = 1. As mentioned in conclusion of measurements – one attempt was chosen to receive and synchronize.

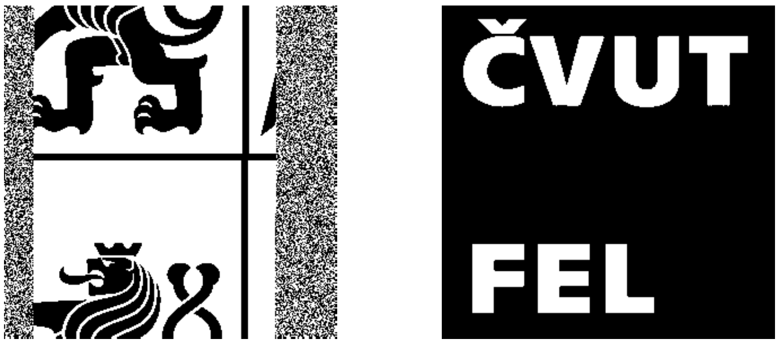

Figure 4.2. Example of the most successfully received pictures from the over-the-air base-level simulation.

From the received pictures in the figure 4.2 we can see, that decoding was either successful with no errors or fully unsuccessful with around 50 % of error bits. It is because of antennas were close to each other and high Tx and Rx gains were used, i.e. there was high SNR in the channel.

In the figure 4.3 we can see some less successfully transmitted pictures. From this trial, we can see, that some kind of a “buffer-overlap” can occur.

4.2 Butterfly network simulation

In the previous section a base-level simulation has been developed. The task of this section is to illustrate a way how to simulate a complex network using the base-level simulation skeleton. But first, we will examine H-frame decoding.

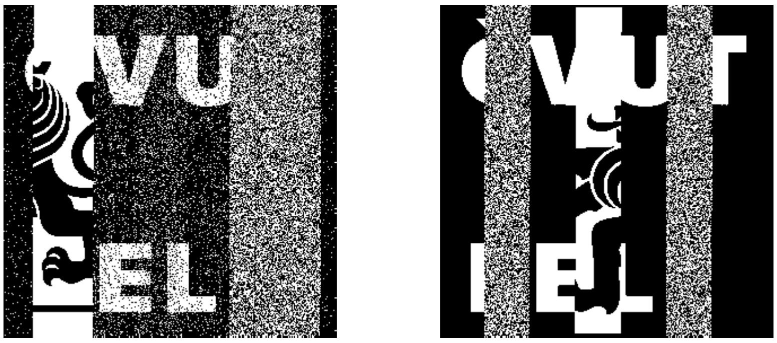

Figure 4.3. Example of less successfully received pictures from the over-the-air base-level simulation.

For LDPC decoder we need LLR. In case of H-frame, LLR depends on channel attenuation in both subchannels and a noise variance. So, we need to extract the channel estimation part from H-frame for both sources. Then, we will estimate channel parameters, demodulate payload and compute LLR. In HMAC stage, transmitters need to be synchronized well. If they are not synchronized, an additional processing is required to handle it. To look for the channel estimation part of H-frame of the second source, the function `matlab_files\func\estim_delay.m` has been created. It takes an `x` – signal corrupted by CWGN and `pilots` – a look-for sequence. The function performs correlation of these vectors and returns position of maximum magnitude of the correlation. It supports multiple corrupted signals – column-wise structure of `x` is assumed. `pilots` can be a column-vector or a matrix with the same number of columns as `x`.

Examination of what happens when sources in HMAC stage are not fully synchronized and how it affects decoding, was done in the script `matlab_files\testFiles \testHMAC_synchChanEst.m` First, there is a demodulation error examined. BPSK symbols for both sources are created and modulated to complex envelopes. Expected H-symbols (`s_symbs`) are computed. Next, a model of received signal is constructed – H-channel model without noise is applied to unsynchronized envelope samples. Envelopes are unsynchronized by `k0` samples. Further, an ad-hoc methods to retrieve expected H-symbols are tried. First method is an application of modified demodulation pulse `pulse_f` to H-signal. `pulse_f` is constructed as a mean of original and shifted version of modulation pulse: $f[n] = (g[n] + g[n-k0])/2$. Demodulation starts from the first sample of composed H-signal. Second method is an application of original (de)modulation pulse `pulse_g` to H-signal, but starting from 1st up to `k0 +1`-th H-signal sample. Demodulated H-symbols are compared to the expected H-symbols using mean magnitude square error (MSE). MSEs are printed to the command line. For the shift `k0` $= 1$, MSE is minimal for demodulation by `pulse_f`. For other shifts `k0` the MSE is minimal for demodulation with `pulse_g` with displaced H-signal beginning. In the table 4.1 we see the option of start index for demodulation of H-signal using `pulse_g`, that minimizes MSE. There were 4 samples per symbol used. Taking into account only shifts 0 to 4, a formula for start index is `ceil( (k0 + 1) /2 )` (we assume MATLAB indexing – starting from 1).

k0	start index	start index -1
0	1	0
1	1	0
2	2	1
3	2	1
4	3	2
5	4	3

Table 4.1. Start index in H-signal for demodulation by `pulse_g`, that minimizes MSE for different shifts `k0`.

For simplicity, the demodulation of H-signal is done using the pulse `pulse_g` together with adapted start index. In next part we will use a difference from original start index – displayed in the third column of 4.1 and computed as `floor( k0 /2 )`.

In the next part of the script there is overall process of construction and demodulation of H-frame examined. There are auxiliary objects created. It is possible to select also negative `k0`. H-frame is composed and inserted to a longer zero-vector. Finally, noise is added. Then, synchronization is performed and an augmented frame is acquired. Further, there are channel estimation parts extracted. Position of the second channel estimation sequence is estimated using function `estim_delay`. Next, `k0_est` estimate is computed and auxiliary difference `k0_half_est = sign(k0_est)*floor(abs(k0_est)/2);` is obtained. The difference is used to shift start and end index of payload envelope and second channel estimation part. Script proceeds by the estimation of channel parameters followed by demodulation of envelope, computation of LLR and final code-word decoding. There is an information printout about a number of error bits at the end. The function `matlab_files\func \prepare_envSamp.m` is used in the script. It is almost identical to the version with `_short` suffix, with a difference, that it performs LDPC encoding. Findings about demodulation of H-frame from the script are applied in the function `matlab_files\func\processFrame_short.m`.

In the remaining part of the section we will focus on simulation of complex networks using base-level simulation skeleton. Whole main loop remains the same, only objects are used in a circular manner and several variables cycle through predefined values. A code for the change of currently active objects and variables is added to the main loop of base-level simulation scripts. To simulation a complex network using base-level simulation, we need to design, resp. to fit transmission and reception into a one2one manner – into "pairs". For Butterfly network 1.1, the pairs are proposed as in the in figure 4.4. There is used a multiple source for S_A and S_B. And a multiple destination for D_A and D_B. The relay, functioning as a destination is in a different pair than the relay functioning as a source to enable transmission of subframes. Relay needs to wait to receive whole H-frame, then it can decode it, again encode and transmit it.

One pair of Tx and Rx works until one codeword is exchanged. Then a switch to the second pair should be done after transmitters and receivers in the pair have exchanged their one code-word, the simulation should switch again to the first pair. Recall, that the main-loop of the base-level simulation consists of 4 parts: 1) data preparation, 2) transmission, 3) reception of data from the other script and 4) processing of the received data. There will be a code for a change of current objects and variables inserted. An example of pair exchange for Butterfly network is depicted in the schema 4.5. There are 2 subframes in one LDPC frame used in the example schema. So, every transmitter transmits 2 times before it changes to a next transmitter, and also

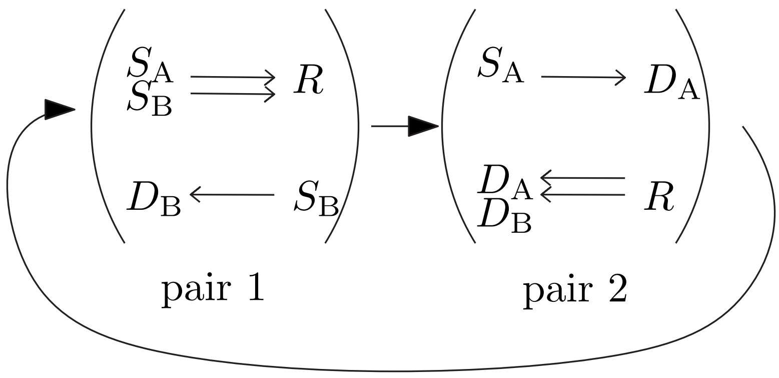

Figure 4.4. Proposed pairs of Tx and Rx for Butterfly network simulation using base-level simulation skeleton.

every receiver receives 2 times before it switches to a next receiver. To emphasize relay functioning as a destination the D_R is used, similarly S_R is used to emphasize the relay working as a source.

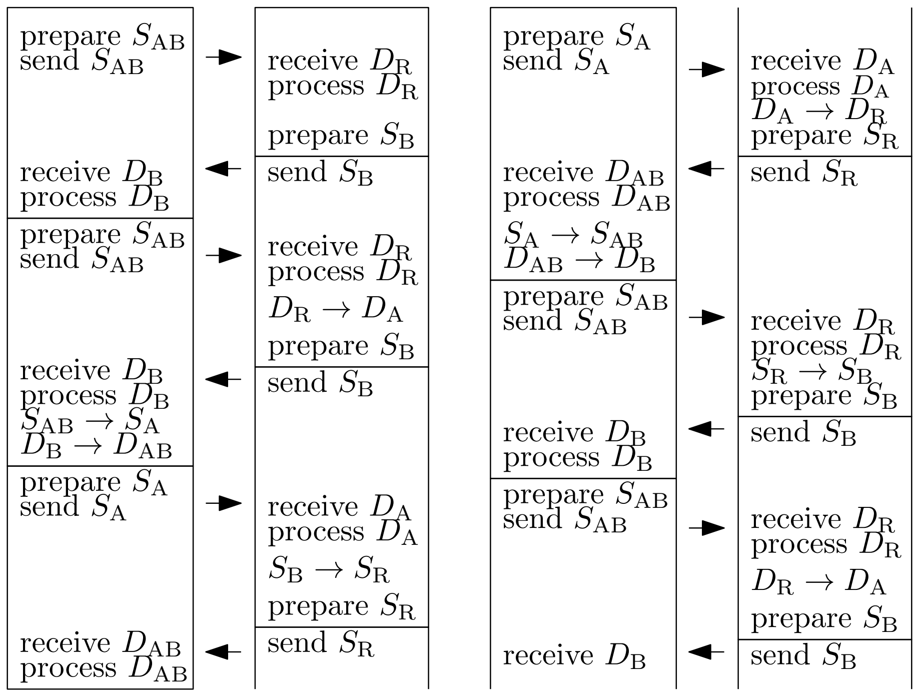
prepare S_{AB} send S_{AB}
receive D_R process D_R
prepare S_B
receive D_B process D_B
send S_B
prepare S_{AB} send S_{AB}
receive D_R process D_R $D_R \to D_A$ prepare S_B
receive D_B process D_B $S_{AB} \to S_A$ $D_B \to D_{AB}$
send S_B
prepare S_A send S_A
receive D_A process D_A $S_B \to S_R$ prepare S_R
receive D_{AB} process D_{AB}
send S_R
prepare S_A send S_A
receive D_A process D_A $D_A \to D_R$ prepare S_R
receive D_{AB} process D_{AB} $S_A \to S_{AB}$ $D_{AB} \to D_B$
send S_R
prepare S_{AB} send S_{AB}
receive D_R process D_R $S_R \to S_B$ prepare S_B
receive D_B process D_B
send S_B
prepare S_{AB} send S_{AB}
receive D_R process D_R $D_R \to D_A$ prepare S_B
receive D_B
send S_B

Figure 4.5. An example of a change of pairs of transmitters and receivers in a pair for Butterfly network simulation using 2 subframes in one LDPC frame.

On the left, there is process 1. On the right, there is process 2. One iteration of main loop is visualized by rectangle around corresponding commands. If we mark number of subframe necessary for one whole LDPC frame by `N_subFr`, we can say that, in process 1, there is a change of receiver and transmitter every k * `N_subFr`-th iteration of the main loop; where `k` is a natural number. In process 2, there is a change of receiver every k * `N_subFr`-th iteration of the main loop and transmitter changes in the next main-loop-iteration after the change of receiver.

Files for simulation of a complex network are these: `matlab_files\testPair \tCh_short[1|2]_ack_whole.m`. There is a function `matlab_files\func \get-PicXOR.m` used in two ways. We will briefly describe its purpose. The function computes XOR function from quantized pictures. In the function, there is `varargin` input parameter, that enables to enter multiple picture files to compute XOR of. Input files are read and quantized according to the half of pixel-range. The quantization to logical values is done in the local function `quantizeIm`. Effectively, only first `N_rows * N_cols` picture pixels are taken from the linearized picture matrix. If necessary, pictures are periodically extended. The extension is done by the local function `extentPic`. Finally, XOR function of all pictures is computed and returned as a logical matrix of size `N_rows x N_cols`. The function is often used with one picture as a parameter. In this case, the picture is quantized and first `N_rows * N_cols` picture pixels are returned.

In the scripts, a pair change is implemented according to the figure 4.4. Auxiliary cell arrays are constructed. There are cell array of bitSources `bsObjs`, bitEval objects `beObjs`, transmitters `TxObjs`, receivers `RxObjs`, frame synchronizers `frSynchObjs`, pilot envelopes samples `pilots` and channel attenuations `h_atten`. A little note on objects. In MATLAB there are two types of objects: [1] value objects and handle objects. When copying value-objects there is a new separate object created. The new object is independent on the original one. When creating handle objects a handle is returned. It is a reference to the object. When copying the handle, only a handle value is copied, no new object is created. This property of handle objects is used. All `matlab.System` objects are derived from `handle` class. Also the relay class `bitRelay2` is derived from `handle` class. We collect handles to an existing objects in cell arrays and we cycle through them. The cycling is directed by an index `idx_pair` and performed by an auxiliary function `matlab_files\func \getIdxthCell.m`. The function takes an index `idx` and a variable number of cell arrays and returns `idx`-th cell value from each cell array. According to the figure 4.5, a code for pair change is inserted to the main loop in both scripts. In the script 1, source and destination are changed together, when a whole frame for LDPC decoding is transmitted – i.e. the condition `if mod(i * cs.N_PL, cs.N_ldpc) == 0` holds. The code is the last part of the main loop. In the script 2, destination is changed earlier than source by one loop iteration. As in the script 1, destination changes when the condition `if mod(i * cs.N_PL, cs.N_ldpc) == 0` holds. Together with the destination update the cycling index `idx_pair` is updated. Source is changed in the next loop iteration, when the condition `if i>1 && mod((i-1) * cs.N_PL, cs.N_ldpc) == 0` holds. The source-update is done using the current `idx_pair`.

Next, we will present simulation results. In the figure 4.6 we can see pictures received in a direct transmission from source to destination. On the left, there is a picture received in destination B. On the right, there is a picture received in destination A.

Effectively they are the same as in base-level simulation 4.1. They are only periodically extended a little. The extension is done so to enable construction of a natural number of data-words for LDPC encoding and not overwrite beginning of pictures when receiving. In the figure 4.7 we can see pictures received in BC stage and MAC stage. On the left, there is a picture received in BC stage in both destinations. On the right, there is a picture received in MAC stage in relay.

Simulation settings was as follows:

- MAC channel attenuations: $h_\mathrm{A} = 0.97\mathrm{e}^{\mathrm{j}2.3}$, $h_\mathrm{B} = 0.94\mathrm{e}^{\mathrm{j}1.8}$. Noise variance: $\sigma_w^2 = 0.9$.
- $S_\mathrm{A} \rightarrow D_\mathrm{A}$ channel. Attenuation: $h_\mathrm{AA} = 0.92\mathrm{e}^{\mathrm{j}1.9}$. Noise variance: $\sigma_w^2 = 0.9$.
- BC stage attenuation (the same for both subchannels): $h_\mathrm{BCstage} = 0.97\mathrm{e}^{\mathrm{j}0.3}$. Noise variance: $\sigma_w^2 = 0.85$.
- $S_\mathrm{B} \rightarrow D_\mathrm{B}$ channel. Attenuation: $h_\mathrm{BB} = 0.92\mathrm{e}^{\mathrm{j}1.2}$. Noise variance: $\sigma_w^2 = 0.85$.

[1] `https://ch.mathworks.com/help/matlab/matlab_oop/comparing-handle-and-value-classes.html`

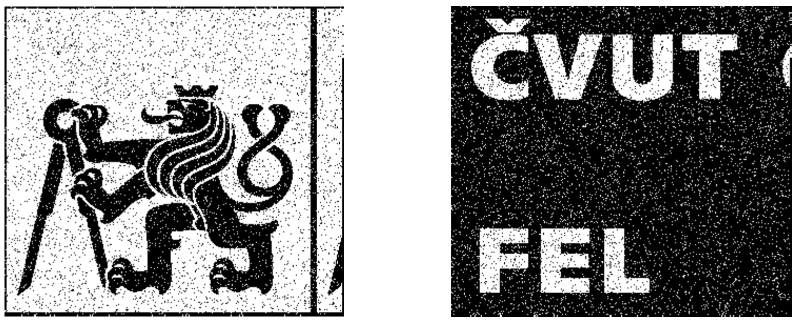

Figure 4.6. An example of received pictures resulted from the Butterfly network simulation run. There in a received picture in destination B directly from source B, on the left. On the right, there is a received picture in destination A directly from source A.

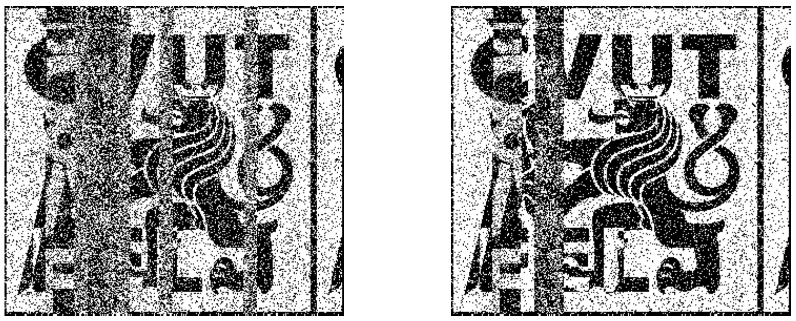

Figure 4.7. An example of received pictures resulted from the Butterfly network simulation run. There in a received picture in destination A,B from the relay, on the left. On the right, there is a received picture in relay from sources A and B in HMAC stage.

- Two subframes per whole LDPC frame were used. I.e. payload of one frame consisted of 8100 code-word bits.

  Error rate of received pictures:

- Destination B, directly from source B (4.6, left). 5.9 % of error bits.
- Destination A, directly from source A (4.6, right). 7.21 % of error bits.
- Relay, from HMAC stage (4.7, right). 17.9 % of error bits.
- Destinations A and B in BC stage (4.7, left). 26.5 % of error bits.

The error is computed with respect to the expected pictures. In relay we expect to decode XOR or source images. In destinations of BC stage we also expect to receive XOR of source images. From the error rate enumeration, we can observe that in one to one transmission there is approximately 6.5 % of error bits after decoding. We can say, that error rate of BC stage reception (26.5 %) is the error rate of relay reception (17.9 %) roughly increased by this rate. Further, we can notice incorrectly decoded codeword in relay picture (4.7, right) – there is one, more or less, black strip. The one strip before the black one seems to be inverted and shifted down. This phenomenon may be caused by simulation imperfections, specifically a concurrence issue.

Finally, a version of the simulation for software radios was done. The simulation files are `matlab_files\testPair \tCh_short[1|2]_ack_wholeHW.m`. Receivers and transmitter have been changed for SDRuReceiver and SDRuTransmitter objects. In the script 1, you can notice the creation of SDRu objects operating two physical devices. They are created with multiple IP addresses and channel mapping:

```
  'IPAddress',[cs.ip.ip1, ', ' cs.ip.ip4],...
  'ChannelMapping',[1,2],...
```

There is a burst mode transmission used and number of frames in burst is set to 1. To ensure long frames are transmitted as a unit. An error occurred when switching from multiple-device transmitter $S_{\mathrm{A,B}}$ to single-device transmitter S_{A}. The objects were created using the same device (with IP address ip1). MATLAB complained that, it cannot use single-device transmitter because it is currently assigned to another active object. So at least, the most interesting part of the simulation was performed – the HMAC stage. The most successful results of received and decoded images in relay from HMAC stage are in the figure 4.8.

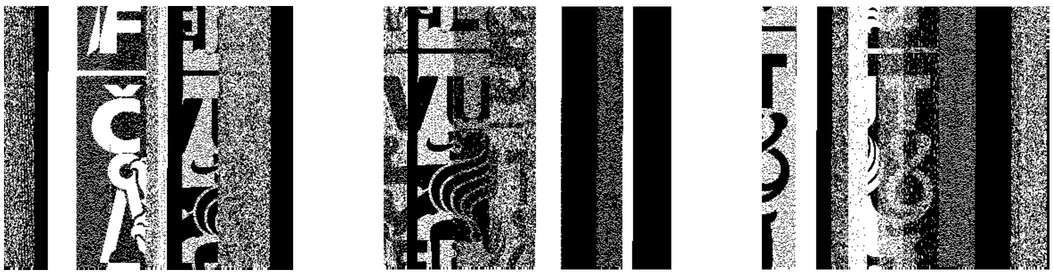

Figure 4.8. An example of the most successfully received and decoded images in HMAC stage from real over-the-air simulation.

For completeness we add also the most successfully received pictures in the other process. It is the picture from destination B. The best received images are in the figure 4.9.

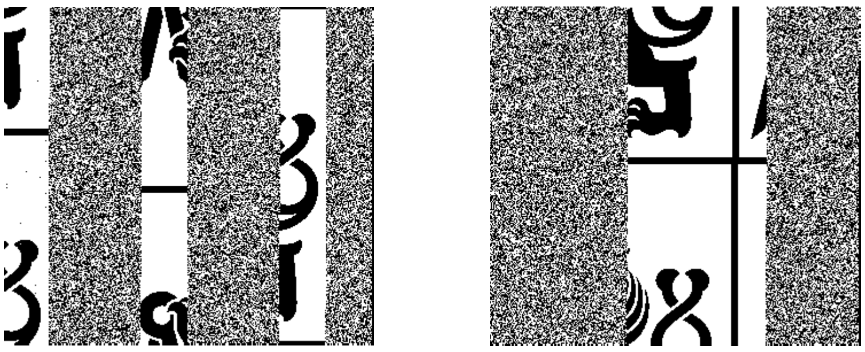

Figure 4.9. An example of the most successfully received and decoded images from destination B, directly receiving from source B, from real over-the-air simulation.

The simulation was set to send and receive a whole LDPC frame at once. In the pictures, received and decoded by relay (4.8), we can see, that some frames were decoded with complete decoder failure – there are fully black and fully white strips. Some strips are at least grainy – decoder returned an alternating code-word sequence. One strip in the left picture expose a bit-inversion. Finally, some strips indicate a more-or-less correct decoding.

4.3 Conclusion

Even though the simulations didn't work as was expected, some correctly received frames confirmed the correctness of the derived formulae for synchronization, channel estimation and computation of the LLR. For the future simulations using software radios, I would suggest not to use Tx and Rx mixed in one process. I think, when there was a switch from Tx to Rx in one process, Rx received at least a bit of transmitted signal. The better option would be to distribute all the Tx to one process and to use only one Rx in the second process. ACK confirmation and relay simulation could be done with help of a shared file.

The goal of this thesis was to get acquainted with fundamentals of WPNC and to implement a MATLAB simulation of a Butterfly network, that would serve as a benchmark for the over-the-air experiments. In the thesis :

- There are estimators for H-MAC and H-BC channel derived.
- CRLB for the derived estimators is evaluated and compared to the computed variances.
- There are functions and objects for composition of the frame envelope developed. The envelope then can be directly transmitted using comm.SDRuTransmitter object.
- Functions and objects for processing of a frame envelope were designed.
- Transmission properties of the radios were examined.
- A basic over-the-air simulation was executed and resulted pictures were presented.

All the code was developed to be able to run on MATLAB R2014a and MATLAB R2017a.

Appendix A
Abbreviations

ACK	■	Acknowledgement
AF	■	Amplify and Forward
API	■	Application Programming Interface
ARQ	■	Automatic Repeat Request
AWGN	■	Additive White Gaussian Noise
BC	■	Broadcast Channel
BPSK	■	Binary Phase Shift Keying
CE	■	Convex Envelope
CRLB	■	Cramér-Rao Lower Bound
CWGN	■	Complex Additive White Gaussian Noise
DVB-S2	■	Digital Video Broadcasting - Satellite - Second Generation
FIR	■	Finite Impulse Response
H-	■	Hierarchical-
H-SODEM	■	Hierarchical Soft-Output Demodulator
HDF	■	Hierarchical Decode and Forward
HNC	■	Hierarchical Network Code
IH	■	Isomorphic Hierarchical
JDF	■	Joint Decode and Forward
LDPC	■	Low Density Parity Check
LLR	■	Log-Likelihood Ratio
MAC	■	Multiple Access Channel
ML	■	Maximum Likelihood
MSE	■	Mean Square Error
NCM	■	Network Coded Modulation
PDF	■	Probability Density Function
PHY	■	Physical
PN	■	Pseudorandom
REC	■	Rectangular
RRC	■	Root Raised Cosine
Rx	■	Receiver
SISO	■	Single Input Single Output
SR	■	Software radio
Tx	■	Transmitter
WPNC	■	Wireless Physical layer Network Coding
XOR	■	Exclusive OR

Appendix B
Mathematical Derivations

The appendix contains mathematical derivations used in the text.

B.1 Bessel J_0 function

The J_0 function is given using polynomial expansion as [8] (eq. 78):

$$J_0(z) = \sum_{k=0}^{\infty} (-1)^k \frac{(z^2/4)^k}{(k!)^2}. \tag{1}$$

We will point out that for $z \in \mathbb{R}$ the function $J_0(-\mathrm{j}z) = I_0(z)$ [9] is *increasing*. Direct substitution to the formula:

$$J_0(-\mathrm{j}z) = \sum_{k=0}^{\infty} (-1)^k \frac{(-1)^k (z^2/4)^2}{(k!)^2} = \sum_{k=0}^{\infty} \frac{(z^2/4.)^k}{(k!)^2}. \tag{2}$$

Clearly, there are only positive expansion coefficients, so the function $J_0(-\mathrm{j}z)$ for $z \in \mathbb{R}$ is *increasing* on $\mathbb{R}$.

For the J_0 function holds [8] (eq. 79):

$$J_0(z) = \frac{1}{\pi} \int_0^{\pi} \mathrm{e}^{\mathrm{j}z\cos\theta}\, \mathrm{d}\theta. \tag{3}$$

We will show another two identities:

$$\begin{aligned} J_0(z) &= \frac{1}{\pi} \int_0^{\pi} \mathrm{e}^{\mathrm{j}z\cos\theta}\, \mathrm{d}\theta \overset{(a)}{=} \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{j}z\cos\theta}\, \mathrm{d}\theta, \\ &\overset{(b)}{=} \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathrm{e}^{\mathrm{j}z\cos(\theta+\gamma)}\, \mathrm{d}\theta, \quad \forall \gamma \in \mathbb{R}. \\ \text{Recall } J_0(-\mathrm{j}z) &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathrm{e}^{z\cos(\theta+\gamma)}\, \mathrm{d}\theta \text{ is increasing (2) on } \mathbb{R}, \forall z \in \mathbb{R}. \end{aligned} \tag{4}$$

(a) – even symmetry of $\cos\theta$ function. (b) – integration of a periodic function over one period doesn't depend on the interval shift.

Now

B.2 Trigonometric identity 1

$$A\cos(\alpha) + B\sin(\alpha) = C\cos(\alpha+\beta) \tag{5}$$

We will determine C and β.

$$\begin{aligned} C\cos(\alpha+\beta) &= C\cos(\alpha)\cos(\beta) - C\sin(\alpha)\sin(\beta), \\ A &= C\cos(\beta), \quad B = -C\sin(\beta), \\ C &= \sqrt{A^2+B^2}, \quad \beta = \arctan(-B/A) + k\cdot\pi, \quad k \in \mathbb{Z} \end{aligned} \tag{6}$$

Note, equation (5) holds for every k in the expression for β (6).

B.3 Expression with a scalar product 1

We will evaluate the derivative of $|\langle x(t), s(t-\tau)\rangle|$ w.r.t. τ The shortcut $\mathrm{sp} = \langle x(t), s(t-\tau)\rangle$ will be used.

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\tau}|\langle x(t), s(t-\tau)\rangle| &= \frac{\mathrm{d}}{\mathrm{d}\tau}\sqrt{\mathrm{sp}\cdot\mathrm{sp}^*},\\
&= \frac{1}{2|\mathrm{sp}|}\left[\langle x(t), s'(t-\tau)\rangle(-1)\cdot\mathrm{sp}^* + \mathrm{sp}\cdot\langle x(t), s'(t-\tau)\rangle(-1)\right],\\
&= -\frac{2}{2}\Re\{\langle x(t), s'(t-\tau)\rangle\frac{\mathrm{sp}^*}{|\mathrm{sp}|}\},\\
&= -|\langle x(t), s'(t-\tau)\rangle|\Re\{\mathrm{e}^{\mathrm{j}\arg\langle x(t),s'(t-\tau)\rangle}\mathrm{e}^{-\mathrm{j}\arg\mathrm{sp}}\},\\
&= -|\langle x(t), s'(t-\tau)\rangle|\cos\left(\arg\langle x(t), s'(t-\tau)\rangle - \arg\mathrm{sp}\right). \qquad (7)
\end{aligned}$$

B.4 Expression with a scalar product 2

Sometimes it's necessary to evaluate s scalar product in L^2 space in which one of the vectors is a derivative of a function. We would like to use samples of original vectors to evaluate it. In the derivation there will be used Fourier transform $\mathbb{F}$ of form: $\mathbb{F}\{a(t)\}(f) = \int_{\infty}^{\infty} a(t)\exp(-\mathrm{j}2\pi ft)\,\mathrm{d}t$, $\mathrm{sinc}(t) = \sin(\pi t)/(\pi t)$ and $\mathrm{rect}(t) = \mathrm{H}(t+1/2) - \mathrm{H}(t-1/2)$ functions ($\mathrm{H}(t)$ is Heaviside function). Assign $\zeta(t) = \mathrm{sinc}(t/T_\mathrm{s})$ (T_s – sampling period).

$$\begin{aligned}
\langle a(t), b'(t)\rangle &= \int_{-\infty}^{\infty} a(t)b'^*(t)\,\mathrm{d}t = \int_{-\infty}^{\infty}\sum_n a_n\zeta(t-nT_\mathrm{s})\sum_m b_m^*\zeta'^*(t-mT_\mathrm{s})\,\mathrm{d}t,\\
&= \sum_n a_n\sum_m b_m^*\left\langle\zeta(t-nT_\mathrm{s}), \zeta'^*(t-mT_\mathrm{s})\right\rangle,\\
&= \sum_n a_n\sum_{m\neq n} b_m^*\frac{(-1)^{n-m}}{n-m}, \neq 0\\
&= \sum_n a_n\sum_{q\neq 0} b_{n-q}^*\frac{(-1)^q}{q}. \qquad (8)
\end{aligned}$$

$$\begin{aligned}
\left\langle\zeta(t-nT_\mathrm{s}), \zeta'^*(t-mT_\mathrm{s})\right\rangle &\overset{(a)}{=} \left\langle\mathrm{e}^{-\mathrm{j}2\pi fnT_\mathrm{s}}\cdot T_\mathrm{s}\mathrm{rect}(T_\mathrm{s}f), \mathrm{j}2\pi f\mathrm{e}^{-\mathrm{j}2\pi fmT_\mathrm{s}}\cdot T_\mathrm{s}\mathrm{rect}(T_\mathrm{s}f)\right\rangle,\\
&= (-\mathrm{j})2\pi T_\mathrm{s}^2\int_{-1/(2T_\mathrm{s})}^{1/(2T_\mathrm{s})} f\mathrm{e}^{-\mathrm{j}2\pi fT_\mathrm{s}(n-m)}\,\mathrm{d}f,\\
&\overset{(b)}{=} -\mathrm{j}2\pi T_\mathrm{s}^2\left[\frac{\mathrm{e}^{-\mathrm{j}2\pi fT_\mathrm{s}(n-m)}}{-\mathrm{j}2\pi T_\mathrm{s}(n-m)}\left(f - \frac{1}{-\mathrm{j}2\pi T_\mathrm{s}(n-m)}\right)\right]_{-1/(2T_\mathrm{s})}^{1/(2T_\mathrm{s})},\\
&= \frac{1}{n-m}\left[\cos(\pi(n-m)) - \frac{1}{\pi(n-m)}\sin(\pi(n-m))\right],\\
&= \frac{(-1)^{n-m}}{n-m}
\end{aligned}$$

(a) – Parseval theorem applied. (b) – assume $n-m\neq 0$; for $n=m$ the integral is zero.

Appendix C
Included Files

```
matlab_files
|---- func
    |-- addAckMark.m
    |-- demodulate_symbs.m
    |-- dvbs2ldpc_custom.m
    |-- estim_delay.m
    |-- estim_chanParams.m
    |-- genNoise.m
    |-- getBitsToSend_short.m
    |-- getIdxthCell.m
    |-- getPicXOR.m
    |-- getRandZeroMean.m
    |-- initObjs.m
    |-- isFrameAcknoledged.m
    |-- llr4ldpcDec.m
    |-- modulate_symbs.m
    |-- nodes_config.m
    |-- nodes_config_matFile.m
    |-- nodes_config_matFile_short.m
    |-- nodes_config_short.m
    |-- nodes_config_static.mat
    |-- nodes_config_static_short.mat
    |-- prepAppZeros.m
    |-- prepare_envSamp.m
    |-- prepare_envSamp_short.m
    |-- printInfo.m
    |-- processFrame_short.m
    |-- srrc.m
    |-- waitForFrame.m
|---- syObjs
    |-- bitEval.m
    |-- bitEval_multi.m
    |-- bitRelay2.m
    |-- bitSource.m
    |-- bitSource_multi.m
    |-- frameSynchEnv.m
    |-- RxEmul.m
    |-- TxEmul.m
|---- testFiles
    |-- eval_CRLB.m
    |-- testBitEval.m
    |-- testBitEval_multi.m
    |-- testBitRelay.m
    |-- testBitSource.m
```

```
   |-- testConvolution.m
   |-- testFrameSynchEnv.m
   |-- testHMAC_synchChanEst.m
   |-- testLDPCDecoder.m
   |-- testRx.m
   |-- testRxEmul.m
   |-- testTx.m
   |-- testTxEmul.m
|---- testPair
   |-- testChainRx.m
   |-- testChainRx_HW.m
   |-- testChainTx.m
   |-- testChainTx_HW.m
   |-- tCh_short1_ack_v2.m
   |-- tCh_short1_ack_v2HW.m
   |-- tCh_short1_ack_whole.m
   |-- tCh_short1_ack_wholeHW.m
   |-- tCh_short2_ack_v2.m
   |-- tCh_short2_ack_v2HW.m
   |-- tCh_short2_ack_whole.m
   |-- tCh_short2_ack_wholeHW.m
pictures
|-- logo_fel_zkratka_cb.jpg
|-- symbol_cvut_plna_doplnkova_verze_cb.jpg
```

References

[1] Shengli Zhang, Soung Chang Liew, and Patrick P. K. Lam. *Physical Layer Network Coding*. 2007.

[2] Jan Sykora, and Alister Burr. *Wireless Physical Layer Network Coding*. Cambridge University Press, 2018. ISBN 9781107096110.

[3] European Telecommunications Standards Institute. *DVB-S2 Standard*. 2014. `http://www.etsi.org/deliver/etsi_en/302300_302399/30230701/01.04.01_60/en_30230701v010401p.pdf`.

[4] Jan Sykora. *Statistical Signal Processing – Lectures*. 2017.

[5] Scratchapixel. *Monte Carlo Methods in Practice*. `https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-in-practice/monte-carlo-integration`.

[6] Phil Schniter. *Introduction to analog and digital communication – lecture site*. `http://www2.ece.ohio-state.edu/~schniter/ee501/`.

[7] *Stop-and-wait ARQ*. `https://en.wikipedia.org/wiki/Stop-and-wait_ARQ`.

[8] Eric W. Weisstein. *Bessel Function of the First Kind*. 2019. `http://mathworld.wolfram.com/BesselFunctionoftheFirstKind.html`.

[9] Eric W. Weisstein. *Modified Bessel Function of the First Kind*. 2019. `http://mathworld.wolfram.com/ModifiedBesselFunctionoftheFirstKind.html`.