Project proposal for ESA Summer of Code in Space 2015 A VOLK-ified POLAR code implementation in GNU Radio

Johannes Demel
johannes.demel@student.kit.edu

March 26, 2015

Contents

1 Introduction

Claude Shannon and Warren Weaver originally published "The mathematical theory of communication" [1] in 1949. In this book they develop a theoretical concept for channel coding. NASA's Voyager 2 mission used convolutional and Reed-Solomon codes. Later Turbo codes were introduced in order to get closer to the Shannon limit. Lately LDPC codes became popular as a coding standard for communication links. However up until recently no channel codes were known that are symptotically good in a sense stated by Shannon[1]. In 2009 Erdal Arikan first proposed a new method of constructing codes that reach channel capacity for binary symmetric memoryless channels, [2]. These codes are named POLAR codes after the polarization effect they exploit.

GNU Radio includes a framework for channel codes called gr-fec. It provides skeletons for encoders and decoders which provide a common interface for all sorts of codes. Currently only convolutional and Reed-Solomon codes are available in-tree in GNU Radio. Other coding families such as Turbo codes or LDPC codes are available.

As a part of GNU Radio's performance enhancements VOLK provides fast SIMD kernels. Although latest development moves VOLK into its own subproject, it is still an integral part of GNU Radio. Besides a large list of basic functionality also more sophisticated algorithms such as a convolutional codes are implemented. Users profit form VOLK in multiple ways. They can use faster standard GNU Radio blocks and they can call VOLK kernels directly to speed up their own blocks.

The objective of my project is to add POLAR codes to GNU Radio. This includes an optimized encoder and decoder as well as suitable algorithms for channel construction. GNU Radio's gr-fec API will be used to implement an encoder and decoder. A Python implementation will serve as a first reference ahead of the optimized high performance C++ implementation. This will also provide improved unit tests and thus help for better code quality.

2 POLAR codes

POLAR codes consist of three parts, an encoder, a decoder and the so-called channel construction. In this section I will present the basics of all three aspects. In order to use POLAR codes a user chooses a symmetric binary discrete memoryless channel (DMC) W, an error probalility p, a block size N, where N is a power of 2, and a code rate R. Code rate R is given as $R = \frac{K}{N}$ with K being the number of *information bits*. The result of channel construction is a vector with the Bhattacharyya parameters, also called Z parameters, or channel capacities respectively. Then the K channels out of N with smallest Z parameters are chosen to carry *information bits*. All other channels carry *frozen bits* with known values and positions within each block. The created POLAR code is then fed to the encoder and decoder for use in a communication system.

This section is written with respect to [2]. First consider a binary symmetric memoryless channel $W : \mathcal{X} \rightarrow \mathcal{Y}$. Its input alphabet shall always be $\mathcal{X} = \{0, 1\}$. Though

[1]Claude was caught exceeding the Shannon-Limit.

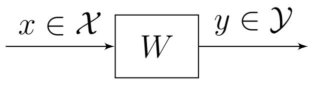
its output alphabet depends on the considered channel and is left arbitrary. In Fig. 1 a generalized channel example and a BSC and a BEC are shown. In case of BSC an error probalility p_e is given which denotes the probalility that a received symbol is unlike the transmitted symbol. A BEC does not receive erroneous symbols but either receives the correct one or a none-symbol indicating an error. A channel's transition propablities are denoted by $W(x|y)$. For further considerations the mutual information

$$I(W) = \sum_{y\in\mathcal{Y}}\sum_{x\in\mathcal{X}} \frac{1}{2} W(y|x) \log \frac{W(y|x)}{\frac{1}{2}W(y|0) + \frac{1}{2}W(y|1)} \tag{1}$$

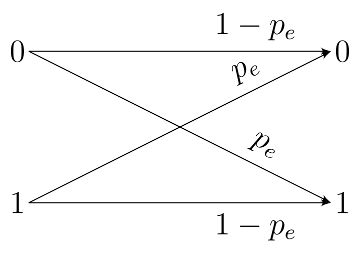
as a means of channel quality and the Bhattacharyya parameter

$$Z(W) = \sum_{y\in\mathcal{Y}} \sqrt{W(y|0)W(y|1)} \tag{2}$$

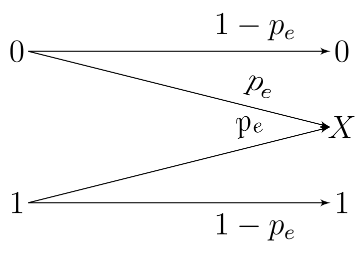
as a measure of reliability are of importance.



(a) generic model, W^1



(b) BSC transition probalilities



(c) BEC transition probalilities

Figure 1: binary channel examples

2.1 Encoder

Fig. 2a shows how a W^2 POLAR channel is contructed. It uses two copies of W and computes $x_0 = u_0 + u_1$ on $GF(2)$ to construct a combined channel. This copy-and-combine process is repeated until the desired power of two number of channels are created and denoted by W^N. Channel combining can be described by 2 matrices.

$$F = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \qquad G_N = B_N F^{\otimes n} \qquad N = 2^n, \quad n \geq 0 \tag{3}$$

Firstly F is a channel combination matrix and $F^{\otimes n}$ is its n-th Kronecker product. Secondly B_N is called *bit-reversal* matrix. B_N reorders the rows in $F^{\otimes n}$ and is comparable to FFT *bit-reversal*. Thus, an encoder performs a matrix multiplication $x_1^N = u_1^N \cdot G_N$ for each code block. Arikan proposes a more efficient encoding theme which has a complexity of $O(N \log(N))$ [2] which I will implement.

2.2 Channel construction

First, channel combining is performed. A POLAR encoder yields $W_N : \mathcal{X}^N \rightarrow \mathcal{Y}^N$ and the transition probalilities $W_N(y_1^N|x_1^N) = W^N(y_1^N|u_1^N G_N)$. The second channel construction step is channel splitting. It is depicted in Fig. 2b. For every u_i of the vector channel, the conditional probalilities $W_N^{(i)}(y_1^N, u_1^{i-1}|u_i)$ are calculated. With this result the channel capacities $I(W_N^{(i)})$ and Bhattacharyya parameters $Z(W_N^{(i)})$ are calculated. The arrows in Fig. 2b indicate the influence of u_{i+1}^N and y_1^N on the conditional probalilities of u_i.

The polarization effect can be observed in Fig. 3a and Fig. 3b. The individual channels tend to 1.0 or 0.0 respectively. Thus they are either perfect channels or noise-only channels. A user chooses a desired rate and selects the best channels for *information bits*. All other channels will transmit *frozen bits*.

Only BEC channels can be calculated efficiently. For other channels optimized algorithms are proposed in [3].

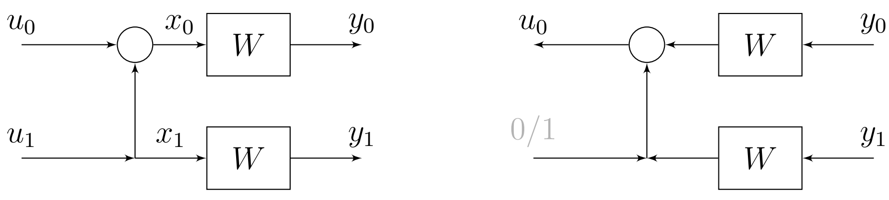
2.3 Decoder

A successive interference cancellation (SIC) decoder is proposed to retrieve transmitted bits at the receiver. This decoder operates at complexity $O(Nlog(N))$. It does not perform well for short block size, thus, a belief propagation (BP) decoder is also proposed by Arikan. Several improved decoders are proposed in [4] and [5] which further improve decoding performance.

3 Deliverables

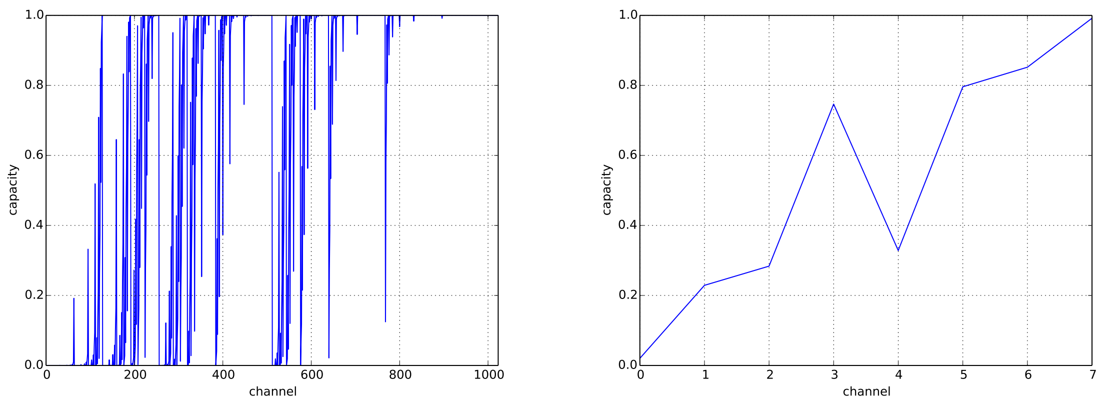
As an ESA Summer of Code in Space project I aim at contributing to GNU Radio a number of features. I will outline those features in this section.

Integration GNU Radio provides the gr-fec API to easily drop in channel codes. FECAPI will be used to integrate POLAR codes directly into GNU Radio's gr-fec module. Directly integrating POLAR codes should improve compatibility with other parts of the FECAPI. This project will be available under the term of the GNU GPLv3. At the end



(a) W^2 POLAR channel combining (b) W^2 POLAR channel splitting

Figure 2: POLAR channel construction



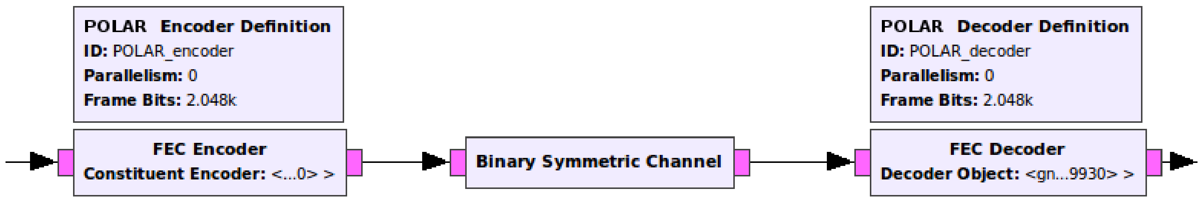
(a) BEC $I(W_N^{(i)})$ for $\eta = 0.3$, $N = 1024$ (b) BSC $I(W_N^{(i)})$ for $p = 0.1$, $N = 8$

Figure 3: Channel capacities for different channels

of the project a pull request for GNU Radio will be prepared in order to make POLAR codes available as easily as possible to every potential user.

Python test code for extensive unit tests shall be created first. This will foster my knowledge with POLAR codes and later serve as a reference in the unit test for the C++ implementation.

Encoders and Decoders are the heart of the signal processing code. They will be coded in C++ and seemlessly integrated into the FECAPI. The integration of encoders and decoders is depicted in Fig. 4. In order to make them as useful as possible they shall be optimized for maximum performance.



POLAR Encoder Definition
ID: POLAR_encoder
Parallelism: 0
Frame Bits: 2.048k

FEC Encoder
Constituent Encoder: <...0> >

Binary Symmetric Channel

POLAR Decoder Definition
ID: POLAR_decoder
Parallelism: 0
Frame Bits: 2.048k

FEC Decoder
Decoder Object: <gn...9930> >

Figure 4: simple POLAR encoder/decoder flowgraph

VOLK kernels will be created that use a CPU's SIMD instruction set for further performance gains. Those kernels will be prepared for merging with the upcoming VOLK project.

Channel construction for large codes will be implemented. Therefore the proposed algorithms [3] will be implemented in Python using NumPy. So users can parameterize there codes and then let the channel construction algorithms do the work to generate desired channel parameters. In order to speed up usage, file save and file read will be provided.

4 Timeline

This is a preliminary timeline to give an overview of the upcoming project stages. It is synchronized to the ESA Summer of Code in Space timeline.

April 30 student application deadline.

May1 - May 31 Community Bonding period 4 weeks, gathering in-depth knowledge of the upcoming project, reading papers, documentation, meetings with mentor for detailed discussions. Choose suitable algorithms for implementation.

June 1 - June 14 Initial Coding 2 weeks, implement encoder, decoder and channel calculation for code construction in Python. Create test suite for C++ implementation.

June 15 - June 29 2 weeks, port encoder and decoder to gr-fec in GNU Radio. Create blocks that can be dropped into the FECAPI framework and make all tests pass.

June 30 - July 5 1 week, optimize encoder and decoder block for maximum throughput

July 6 - July 7 Mid-term wrap up 2 days, prepare and submit results for mid-term evaluation.

July 8 - August 7 4 weeks, implement VOLK kernels for POLAR encoder and decoder.

August 8 - August 23 2 weeks, implement sophisticated channel calculation algorithms to enable use of large vector channels.

August 24 pencils down 1 week, issue pull request and work on integration into GNU Radio main.

August 31 project end project hopefully merged in main.

5 Motivation

I am a Master degree student at Karlsruhe Institute of Technology (KIT), Germany. My major is focused on Communication at Communications Engineering Lab (CEL). Also I am currently a student research assistant at CEL. During my studies and work

I acquired advanced skills in digital signal processing, waveform design and software engineering.

I got in touch with GNU Radio back in 2012 when I started my Bachelor thesis on LTE signal reception with GNU Radio. The outcome of this thesis is available on Github [6]. After I finished my Bachelor thesis I continued to work on it and added more features as well as code improvements. Further outcome of this work are two articles [7] and [8]. During the course of gr-lte development I touched a great variety of different parts of GNU Radio. Firstly adopting and integrating existing blocks into the project. Secondly learning a lot about GNU Radio internals like scheduler specifics, class inheritance structures and VOLK usage. Also, I improved my knowledge to write test code in order to improve code quality.

Besides my studies at KIT I did an internship at Ettus Research in 2013/14, fostering my programming skills and deepening my GNU Radio knowledge. During that time I contributed to Ettus' UHD driver and developed tools for use with GNU Radio [9].

Currently I'm working as a student research assistant at CEL again. My fields of interest are SDR development and system optimization. This includes heavy algorithmic optimization and overall code optimization for maximum throughput and bandwidth capabilities. Recently I contributed to GNU Radio by adding a rename capability to its modtool tool [10].

Besides my experience with GNU Radio and thus with C++ and Python, I also did several smaller projects. That includes closed source projects at university level with Java and Java for Android. I did microcontroller programming in ANSI C and Assembler. This included integration of an IP microcontroller into an existing VHDL project.

Before ESA Summer of Code in Space starts I will have finished my last exams and be fully available for my project. I want to take the opportunity to work on GNU Radio while I am still a student. Also, I aim at using the results of my project as part of my Master thesis. I hope to foster my knowledge and experience with software engineering and channel coding.

During the course of the project, I will have access to CEL's labs and equipment. And Sebastian Koslowski, who is currently with CEL, agreed on mentoring my project. Thus I will always be able to talk to him personally about project progress and pitfalls.

I read GNU Radio's Code of Conduct and understood and agree to it, including the three strikes rule. They are natural to follow and essential for a positive and prospering community. I am looking forward to publish my weekly project updates.

GNU Radio is an amazing tool to rapidly develop new waveforms. I would be more than happy to contribute to it so others can benefit from my work, just as I do from having GNU Radio at hand. Furthermore, I want to use the opportunity to enable development of new channel codes in the hope that one day, they will go to space.

6 Conclusion

Channel codes are a crucial component of every communication system. High performance codes enable reliable high bandwidth data transmission. It is often very difficult to choose an appropriate channel code for a specific use-case. Therefore it is beneficial to have a variety of different implementations at hand, especially for GNU Radio with a strong focus an research and development. Having POLAR codes as an option for channel codes in GNU Radio enables users to evaluate this code for use in many different environments, such as in space. A fast POLAR code implementation for GNU Radio will ease user's lives to conduct tests. POLAR codes are an exciting field for further investigations because they promise to reach Shannon channel capacity.

I would like to take this opportunity to participate in GNU Radio development and contribute to it.

References

[1] C. Shannon, *The mathematical theory of communication.* Urbana: University of Illinois Press, 1998.

[2] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *Information Theory, IEEE Transactions on*, vol. 55, no. 7, pp. 3051–3073, July 2009.

[3] I. Tal and A. Vardy, "How to construct polar codes," *Information Theory, IEEE Transactions on*, vol. 59, no. 10, pp. 6562–6582, 2013.

[4] K. Chen, K. Niu, and J. Lin, "Improved successive cancellation decoding of polar codes," *Communications, IEEE Transactions on*, vol. 61, no. 8, pp. 3100–3107, 2013.

[5] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *Communications Letters, IEEE*, vol. 16, no. 12, pp. 2044–2047, 2012.

[6] "gr-lte Repository," https://github.com/kit-cel/gr-lte, accessed 2015-02-12.

[7] J. Demel, S. Koslowski, and F. Jondral, "A LTE Receiver Framework Implementation in GNU Radio," *SDR-WInnComm-Europe-2013*, June 2013.

[8] ——, "A LTE Receiver Framework Using GNU Radio," *Springer Journal on Signal Processing Systems*, 2015.

[9] "gr-misc Repository," https://github.com/jdemel/gr-misc, accessed 2015-02-12.

[10] "GNU Radio modtool rename contribution," https://github.com/gnuradio/gnuradio/tree/ab6b4208b4e843548d9c8898cbe1102976161663, accessed 2015-02-12.