The Implementation of GNU Radio Blocks for Decoding Long-lasting Frames in Mobile Underwater Acoustic Communications

Jamil Kassem JAMILAHKASSEM@GMAIL.COM
Lebanese International University, Bekaa, Lebanon

Michel Barbeau BARBEAU@SCS.CARELETON.CA
School of Computer Science, Carleton University, Ottawa, ON, Canada, K1S 5B6

Abdel-Mehsen Ahmad ABDELMEHSEN.AHMAD@LIU.EDU.LB
School of Engineering, Lebanese International University, Bekaa, Lebanon

Joaquin Garcia-Alfaro JOAQUIN.GARCIA_ALFARO@TELECOM-SUDPARIS.EU
SAMOVAR, Telecom SudParis, Paris-Saclay University, France

Abstract

We present the implementation of new GNU Radio blocks that support mobile underwater acoustic communications. More specifically, to address the Doppler shift that occurs during the transmission of data frames at a very low data rate. We aim at long distance communications, which require low frequency and extremely narrow bandwidth modulation, and implies weak signals. We build upon our previous works on ad hoc underwater wireless communications, to handle constant or variable (linearly and nonlinearly) Doppler shift patterns. Experimental results are discussed using simulation of underwater autonomous vehicles and underwater wireless sensors. Our main contributions are in the design of the decoder, implemented using the GNU Radio development toolkit.

1. Introduction

Underwater data communications and networking have applications in monitoring and surveillance of coastal waters (Otnes et al., 2008), submarine activity sensors (Otnes et al., 2012), autonomous undersea vehicles (Button et al., 2009), underwater robots (Antonelli, 2014) and submerged airplane locator beacons (Wikipedia, 2018b). We concentrate on low frequency mobile acoustic communications (Decarpigny et al., 1991; Hixson, 2009), i.e., in the range 300 Hz to 3 kHz. In contrast to higher frequencies, Stojanovic has already pointed out that attenuation is lower in that range (Stojanovic, 2007). Hence, there is potential for long distance contacts (Freitag et al., 2015). However, because of the narrow half-power bandwidth at low frequency and long distance, solely extremely low data rates are possible. Furthermore, the relative mobility of a transmitter and a receiver affects the acoustic waves used for underwater communications. This is the Doppler effect. Contrasted with classical electromagnetic communications, it has a significant impact. For example, according to Marage and Mori, the Doppler effect is almost $4\,000$ times greater in sonar than in radar (Marage & Mori, 2010).

In this paper, we present the implementation of four new GNU Radio blocks to support mobile underwater acoustic communications. We focus on the mobile and long distance communication capability, which implies low carrier frequency, weak signal strength and extremely narrow bandwidth modulation. We build upon our previous research (Barbeau, 2017; Ahmad et al., 2017; 2018a;b). With respect to our work published in the last year proceedings of the GNU Radio conference (Barbeau, 2017), this paper expands with new capabilities that enable decoding of data frames subject to a nonlinear Doppler effect. This is a difficult problem with long lasting data frames (111 seconds in our system) because the carrier frequency must be tracked along the symbols making the frame. The frequency may drift following many different and erratic patterns. We approach the problem pragmatically making assumptions about the cause of the Doppler effect. That is, we assume a straight-line trajectory model for mobile communicating vehicles and infer the corresponding effects on the frequency (Ahmad et al., 2018b). We believe that this assumption is reasonable in the context of underwater communications. It is expected that vehicles travel along certain maritime paths and through certain known channels. This paper focuses on the implementation details

Proceedings of the 8^{th} GNU Radio Conference, Copyright 2018 by the authors.

of the GNU Radio blocks comprising a previously developed straight-line trajectory model (Ahmad et al., 2018b).

The paper is structured as follows. Section 2 is about the high level design of the GNU Radio implementation. Section 3 describes the frequency tracking logic. Section 4 demonstrates the execution of our new GNU Radio blocks together with an experimental scenario of underwater acoustic communications. Section 5 concludes the paper.

2. New GNU Radio Blocks

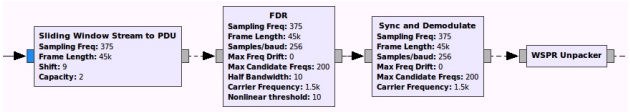

Figure 1. New GNU Radio bocks

The design consists of four new blocks: Sliding Window Stream to PDU, FDR, Sync and Demodulate, and WSPR Unpacker, see Figure 1. Block *Sliding Window Stream to PDU* accepts a continuous stream of time-domain samples. When a temporal window of 120 seconds of signal samples has been buffered, the samples are stored in the payload of a Protocol Data Unit (PDU) that it posted on the output port. The window slides nine seconds forward over the buffer of signal samples. Block *FDR* (Frequency Domain Representation) takes in input such a PDU and constructs a frequency domain representation from the buffered 120 second signal samples. A frequency domain search procedure generates a PDU containing indices of candidate frequencies together with related relevant parameters that include estimated time shift (relative to the sample window start), frequency drift and SNR (Signal to Noise Ratio). Block *Sync and Demodulate* attempts to resolve time delays and demodulate signals at the candidate frequencies. It takes into account the Doppler effect on frequencies. Two Doppler forms are considered: linear and nonlinear.

The linear form is handled considering the maximum frequency shift over the time interval associated with a frame. The nonlinear form is handled making hypotheses about the two communicating objects, i.e., their positions, headings and velocities. The choice of suitable frequency shifts is made exploring a set of plausible candidate values for positions, headings and velocities. These values are chosen such that they likely contain all probable Doppler variations. The linear and nonlinear forms are simultaneously considered to cover all possible variations. The block chooses the form with higher correlation when searching for a frame. If successful, output frame payloads (i.e., packets) are posted, as PDUs.

High-level behavior of the decoder

```
01: for loop #1 // (s)lope search
02:   for loop #2 // (v)elocity search
03:     for loop #3 // (p)osition search
04:       // Find & store candidate tuples as [s, v, p]
05:       // Use triple [s, v, p] for Frequency Tracking:
06:           // Step 1. Energy search:
07:               // Use energy peaks to identify candidate signals
08:           // Step 2. For each candidate signal:
09:               // Try synchronization and demodulation ...
10:               // ... assuming Doppler shifts.
11:       end for
12:     end for
13: end for
```

Figure 2. Pseudocode of the decoder

Figure 2 outlines the search strategy. There are thee embedded *for*-loops. Headings are modelled by straight line on a 2D plane. The slope of each line determines the associated heading. The first *for*-loop explores a range of slopes. The second *for*-loop searches through a domain of plausible velocities. The third *for*-loop scans a number of starting positions, at the beginning of the window of samples. All values of slopes, velocities and positions, are discrete. The body of the loops comprises two steps. Firstly, given a nominal frequency and a triple comprising a slope, a velocity and a position, a frequency tracking procedure finds peaks of energy on the corresponding path in the frequency domain. Secondly, when such a peak of energy exists, then synchronization and demodulation of a frame are attempted.

Figure 3 shows examples of linear and nonlinear Doppler shifts. The figure shows six curves, where the velocity of a transmitter varies from 5 m/s (light yellow curve) to 10 ms/s (darkest curve). For each curve, from start to the 0.25 minute time point the Doppler varies linearly. From 0.25 minute to 3.5 minutes, the Doppler varies nonlinearly. Afterwards, the Doppler effect is present but almost constant.

Figure 4 shows the flow graph of a complete receiver. GNU Radio generates a Python program from it. Each rectangle represents a block, which can be a variable, an instance of a predefined GNU Radio block or one of the aforementioned four new blocks. Top left, the *Audio Source* block interfaces the decoder with a sound card. The sound card bridges the analog underwater acoustic world and dig-

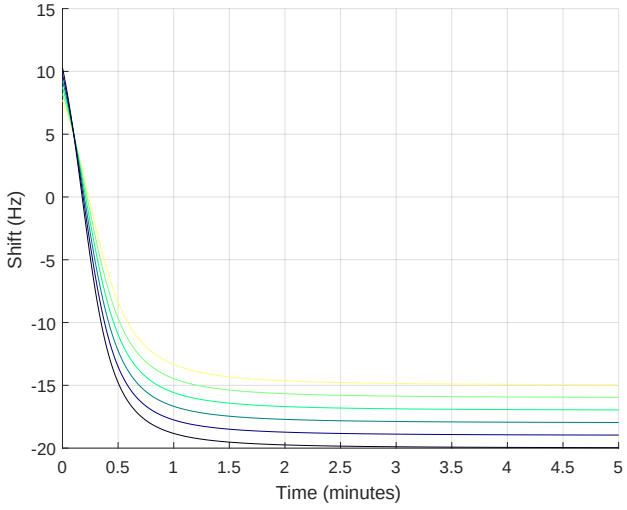

Figure 3. Doppler plots over a five minute interval

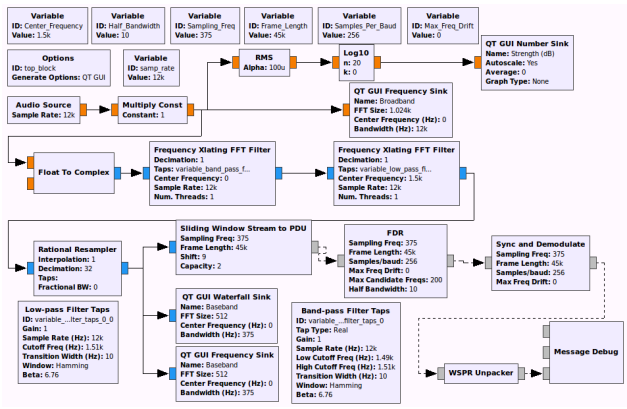

Figure 4. Flow graph of the audio decoder

ital computer environment. Variable *Center_Frequency* determines the nominal frequency of operation, 15 kHz in the example. The audio is sampled at 12 k samples per seconds. Variable *Half_Bandwidth* specifies the effective bandwidth of the decoder. It is the variable value (10 Hz) times two (20 Hz). Control of the effective bandwidth is very important in the underwater environment. In the low frequency range, maritime traffic generates strong noise that can be several orders of magnitude stronger than a weak data signal. Filtered out maritime traffic noise and narrow bandwidth capability greatly increase the performance of the decoder. The first *Frequency Xlating FFT Filter* block achieves this filtering. The second *Frequency Xlating FFT Filter* block down converts the broadband signal to a baseband signal centered at zero Hz. A *Rational Resampler* block decimates the input signal from a high rate to a low rate, by a factor of 32 in this example. Variable *Sampling_Freq* defines the rate at which the analog input is decimated, 375 samples per seconds in this example. In accordance with the Nyquist criterion, this rate also fixes the maximum bandwidth of the decoder. It is less than one half the sampling frequency, i.e., 187 Hz in this case. Following decimation, instances of our new blocks are connected together to decode the input signal into a packet, namely the blocks *Sliding Window Stream to PDU*, *FDR*, *Sync and Demodulate* and *WSPR Unpacker*. Block *WSPR Unpacker* parses packets in the WSPR format (Wikipedia, 2018a) and outputs text content.

3. Frequency Tracking

The movement of vehicles in the sea, and consequently the transmitters and receivers that they embed, generates Doppler frequency shifts in signals. We use frequency modulation. Handling of Doppler shift is required to achieve good results during the decoding process. To mitigate the Doppler effect, we devised a two-step frequency tracking scheme. At the receiver, the tracking scheme takes into account that a carrier frequency can be shifted. This shift may be variable during the reception of a frame. This variation can be linear or nonlinear. This means that it is not sufficient to find where the frequency of a signal is, but in addition the frequency must be tracked during the reception of each frame. In the receiver, the first step consists of an energy search procedure. Energy peaks are identified and used to determine approximate candidate frequencies where valid signals may be present. In a second step, synchronization and demodulation are attempted at each candidate frequency, taking into account the presence of a possibly linear or nonlinear Doppler shift.

3.1. Energy Search

Implemented in the GNU Radio block *FDR*, the energy search aims at finding candidate frequencies where data frames could be present. It takes into account three variables: carrier frequency, frequency drift during data frame reception and data frame start time. The search for candidate frequencies is performed while simultaneously varying the frequency drift as a function of time. The frequency search is coupled with a drift search to mitigate the Doppler effect. The drift search is done by generating a signal with a drift pattern that is compared with the received signal. Correlation of the received signal with a synchronization bit sequence is evaluated. We consider two main frequency drift patterns: linear and nonlinear. The specific nonlinear drift patterns that are examined is configurable. When needed, it can be extended to cover more cases.

A linear Doppler pattern is characterized by a linearly increasing drift starting at zero Hz, at the beginning of a data

frame, up to *maxdrift* Hz at the end of the data frame, where *maxdrift* is a configurable parameter. For example, Figure 5 pictures a linear frequency drift example over an interval of 120 seconds. The carrier frequency is 1.5 kHz and the drift is five Hz. The formal parameter *Max Freq Drift* of the GNU block *FDR* determines the data member *m*axdrift in the implementation. The drift resolution process is iterative. After each iteration, the receiver checks if the current correlation is larger than the previous highest correlated drift (initial maximum correlation is set to zero). The decoder successively evaluates all candidate drifts. It elects the drift yielding the highest correlation.

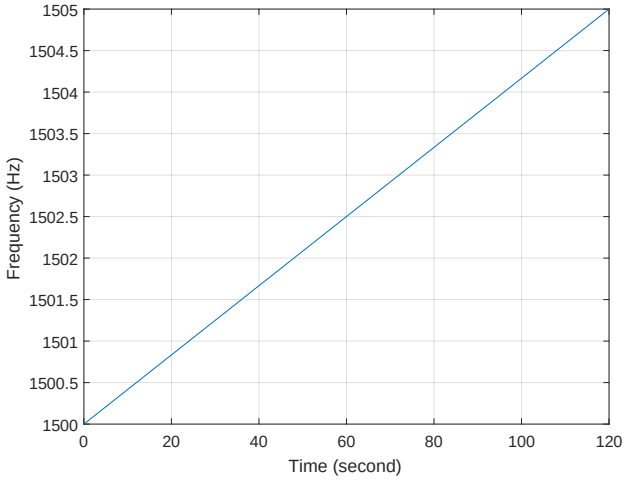

Figure 5. Linear Doppler shift

A nonlinear Doppler effect is produced when one or both communicating vehicles are moving. Moreover, the angles that their velocity vectors are making, with a vector connecting their positions, are variable. Though velocities may be constant, the Doppler effect produces a nonlinear frequency drift due to the variation of these angles. This concept is shown in Figure 6. Represented by a circle and a square, there are two communicating vehicles at constant velocities V_1 and V_2. With respect to a vector connecting their positions, the velocity vectors are at angles α_1 and α_2, shown at time instants T_1 and T_2. As a function of time, α_1 and α_2 are variable. Theoretically, there is an unbounded number of possible nonlinear deviating frequency drift patterns. Besides, we aim at a solution that preserves the capability of the system to deal with extremely weak signals (Barbeau, 2017). We adopted a pragmatic approach. To resolve nonlinear Doppler cases, we make assumptions about possible trajectories of vehicles. In the underwater environment, it is an assumption that is perfectly acceptable since vehicles tend to follow known maritime passages. This is particularly true at the highest velocities. We first compute and explore the time

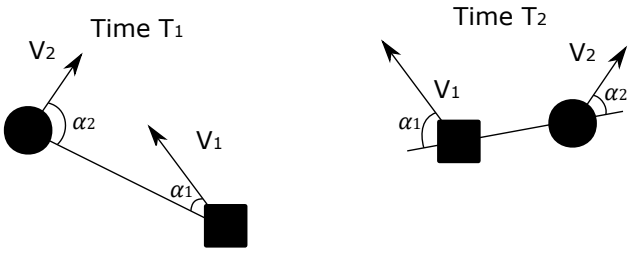

Figure 6. Angles separating objects at different time points

dependent parameters of a range of potential straight-line trajectories, which would model paths through known maritime passages. See Ref. (Ahmad et al., 2018b) for the mathematical details of the model. See also Ref. (Barbeau, 2017) for the signal processing details. Hereafter, we focus on implementation details of the Doppler shift handling aspect. The full source code is available online (github.com/michelbarbeau/gr-uwspr).

All the information about a candidate frequency is stored in an instance of structure `candidate_t`:

```
enum Modes { linear, nonlinear };
typedef struct {
 float freq;
 float snr;
 float sync;
 int shift;
 Modes m_type; // mode type (1: linear, 2: nonlinear)
 union {
  // linear frequency drift model
  mode_linear m_linear;
  // straight−line nonlinear frequency drift model
  mode_nonlinear m_nonlinear;
 };
} candidate_t;
```

In all cases, the candidate is characterized by a nominal frequency (`freq`), a SNR (`snr`), a correlation value (`sync`), and a relative time offset representing a start position for synchronization (`shift`). A union declaration holds the two exclusive types of Doppler effect, defined as modes 1 and 2, with corresponding structures `mode_linear` and `mode_nonlinear`. For the linear case, solely a numerical value characterizing the total amount of frequency drift is stored.

```
typedef struct {
 float drift;
} mode_linear;
```

For the nonlinear case, we store the velocity, slope and start position of the transmitter (vehicle a) and receiver (vehicle b):

```
typedef struct {
```

```
 // vehicle "a"
 double va; // velocity
 double ma; // slope
 int xa, ya; // start position
 // vehicle "b"
 double vb; // velocity
 double mb; // slope
 int xb, yb; // start position
} mode_nonlinear;
```

This information is used to derive the frequency drift while a frame is being received. Referencing the structure `candidate_t`, the straight line trajectory model (Ahmad et al., 2018b) is implemented in class `SLM`. In accordance with the conventions of the GNU Radio framework, the block `FDR` is implemented by the C++ class named `FRD_impl`. By inheritance, the class `FRD_impl` obtains all the properties defined in the class `SLM`:

```
#include "candidate_t.h"
class SLM
{ <Public declarations of class SLM> }
class FDR_impl : public FDR, public SLM
{ ... }
```

Class `SLM` contains the following public declarations:

```
public:
 // returns frequency drift according to the straight line model
 float slmFrequencyDrift(mode_nonlinear m_nl, float cf, float t);
 // generator of parameters for the straight line model
 bool slmGenerator(mode_nonlinear *m_nl);
 // initialize the generator
 void slmGeneratorInit();
```

Given the information about vehicles in motion generating a nonlinear frequency shift `m_nl`, a nominal frequency `cf` and a time instant `t`, method `slmFrequencyDrift` calculates the corresponding drifted frequency. Method `slmGenerator` is a generator of instances of structure `mode_nonlinear`. It is initialized by method `slmGeneratorInit`. Once initialized, method `slmGenerator` consecutively returns all plausible nonlinear cases, one by one.

In class `FRD_impl`, a first analysis of the frequency domain representation identifies peaks of energy, across the corresponding 120 second interval. In the frequency domain, the locations of these peaks become candidate frequencies. Then, a *for*-loop examines every candidate frequency. The parameters of the candidate frequencies are considered tuning the nominal frequency, finding where the start symbol of the frame is in the window and determining a potential frequency drift during the reception of the frame. The parameters are chosen to maximize correlation with a synchronization bit pattern. The synchronization bits are present along the entire frame. Every of the 162 symbols contains a synchronization bit, together with a data bit. The following code illustrates how the generator of nonlinear cases is used:

```
for(j=0; j<npk; j++) { // for each candidate at index j
 // try linear frequency drift
 ...
 // try nonlinear frequency drift
 slmGeneratorInit(); // init generator
 while (slmGenerator(&m_nl)) {
  for (k=0; k<162; k++) { // sum of power over symbols
   // map symbol index to delay in seconds
   t = k * 111 / 162;
   // determine drifted frequency
   ifd = ifr + slmFrequencyDrift(m_nl, carrierfrequency, t);
   ... calculate power at each synchro bit ...
  }
  sync = <total power at positions of synhro bits>
  // synchro bit power versus total power
  sync = ss/pow;
  // more than threshold times current max
  if(sync/candidates[j].sync>threshold)
   candidates[j].shift=<start position>
   candidates[j].freq=<index to tuned frequency>
   candidates[j].sync=sync;
   // a non linear case
   candidates[j].m_type = nonlinear;
   candidates[j].m_nonlinear = m_nl;
  }
 }
}
```

The array `candidates` stores instances of the structure `candidate_t`. The variable `npk` indicates the actual number of instances in the array. The variable `j` is an array index for denoting individual instances. When the *for*-loop is entered, each cell in the array `candidates` identifies a candidate frequency, i.e., `candidates[j].freq`, associated with the presence of an energy peak. The linear case is evaluated first, then the nonlinear case. Hereafter, we focus on the nonlinear case. The method `slmFrequencyDrift` is invoked to initialize the generation process of straight line trajectories. A *while*-loop examines each generated case, represented by the variable `m_nl`. A *for*-loop calculates the power contents at each synchronization bit (there are 162 of them in a 111 second interval). The symbol index `k` is translated to a time offset `t`. The method `slmFrequencyDrift` is called to calculate the frequency drift due to the Doppler effect for the current symbol. It is added to the current tuned frequency `ifr`. The result, i.e., `ifd`, is used to determine and find the power at the synchronization bit. The total power at the positions of the synchronization bits is stored into the variable `sync`. If the ratio of this power versus what has been determined before is higher that a threshold, then this candidate frequency is put in the nonlinear category. The corresponding parameters are stored in the array cell `candidates[j]` for the upcoming steps.

For each of the aforementioned parameters (slope, veloc-

ity and position), a range of plausible values is explored. At each iteration, a new value for one of the parameters is tested. Each combination is successively tested, until they have been exhausted. For each candidate frequency, linear frequency drifts (including a null drift) and nonlinear frequency drifts are explored. The mode yielding the highest correlation with the synchronization bits is retained, together with the parameters determining the drift. To illus-

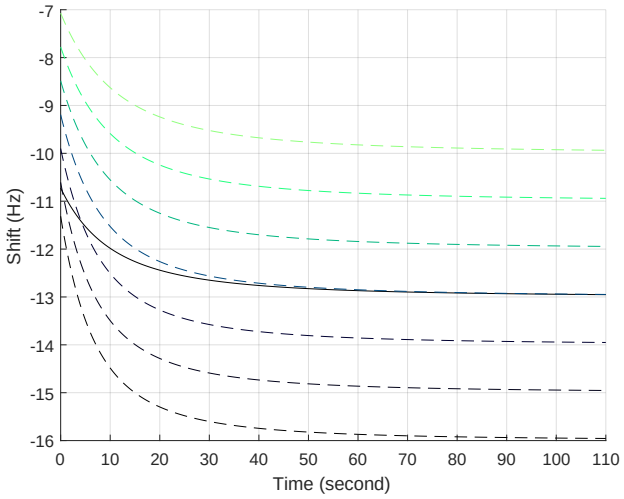

Figure 7. Velocities and timing variations

trate the combination of the drift and time search, let us have a look at a simple example. The Doppler shift of the received signal is represented by the solid black line (cf. Figure 7) while the dashed lines represent the Doppler shift obtained with different velocities that are tested. Notice how the curves do not match. This is because there is also a five second-timing difference from the dashed-curves with the received-signal curve. To solve this issue, we use a time search alongside a frequency drift search. Using the two together yield a good match (cf. Figure 8).

3.2. Sync and Demodulate

Demodulation is implemented in the GNU Radio block *Sync and Demodulate*. The results from the energy search are used to find the bits in the data frames. In accordance with the GNU Radio conventions, the block *Sync and Demodulate* is implemented by the C++ class `sync_and_demodulate_impl`. By inheritance, the class `sync_and_demodulate_impl` obtains all the properties defined in the class `SLM`, together with the properties of the class `Fano` for probabilistic Forward Error Correction (FEC) (Fano, 1963):

```
class sync_and_demodulate_impl :
  public sync_and_demodulate, public SLM, public Fano
{
```

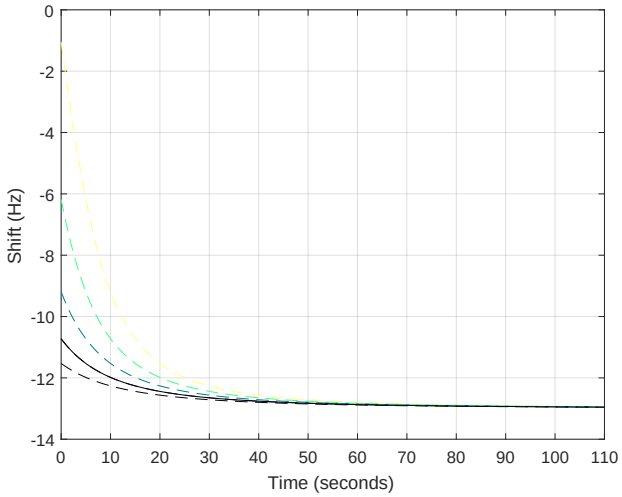

Figure 8. Time synchronization with velocity variations

```
  ...
  // synchronize and demodulate
  void sync_and_demodulate(candidate_t candidate,...);
  ...
}
```

The main logic is implemented in the method `sync_and_demodulate`, which takes as one of its arguments a candidate frequency denoted as `candidate` that has been identified by the GNU Radio block *FDR*. Before demodulating, a refined search is done to improve the estimate for the time offset, where the frame starts in the 120-second window of samples. The logic of this search is similar to the one of the energy search in the block *FDR*, but with refined steps and time intervals:

```
for (i=0; i<162; i++) { // i = frame symbol index
 candidate.m_type==nonlinear);
 switch(candidate.m_type) {
  // linear drift search
  case linear : {
   ... calculate drifted carrier freq ...
   break;
  }
 // nonlinear drift search
 t = i * 111 / 162; // map symbol index to delay in seconds
 case nonlinear : {
  fp = f0 + slmFrequencyDrift(candidate.m_nonlinear,
   carrierfrequency, t);
  break;
 }
}
... non coherent demodulation of symbol ...
```

A *for*-loop examines each of the 162 symbols, indexed by variable `i`. Since the kind of frequency drift has been already determined, only one mode is considered (linear or

nonlinear). According to the mode, the carrier frequency is determined. Focusing on the nonlinear case, the symbol index `i` is translated to a time offset `t`. The method `slmFrequencyDrift` is used again to calculate the frequency drift due to the Doppler effect for the current symbol. It is added to the current tuned frequency `f0`. The result, i.e., `fp`, is used to demodulate the symbol. Noncoherent demodulation of symbols is performed, which means that the signal phase is not recovered (Proakis & Salehi, 2008). We use four frequency-shift keying modulation. For each of the four symbols, the corresponding waveform is generated as function of time using the frequency `fp`. Correlation is calculated with the input signal. The calculated correlation is used to infer a probability for binary values. They are used in input by probabilistic FEC.

4. Example Application

We have conducted several simulations and experiments in various bodies of water, including canals, lakes and coastal waters (Barbeau, 2017). An example application is provided hereafter. One transmitter sends two frames, 110 seconds each at 1.46 baud. The transmitter moves with a constant speed of 8 m/s, leading to a Doppler shift that starts as non-linear and, as the separation distance increases, becomes linear during the transmission of the frames.

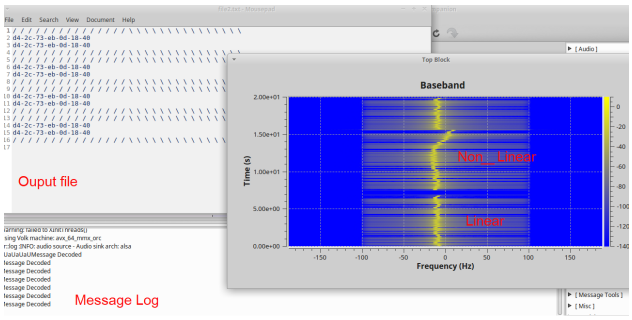

Figure 9. Decoding a two frames sent repeatedly

Figure 9 shows as function of time (vertical axis) the resulting frequency drift (horizontal axis). The decoder is successful at decoding the frames.

In Figure 10, the experiment is repeated, but with AWGN (Additive White Gaussian Noise) such that the resulting SNR is -25 dB. Due to the low SNR, the trace of the signal becomes confused with noise. Although, the decoder is still successful at decoding the frames.

5. Conclusion

We have demonstrated how our four new GNU Radio blocks are capable of decoding frames in an underwater

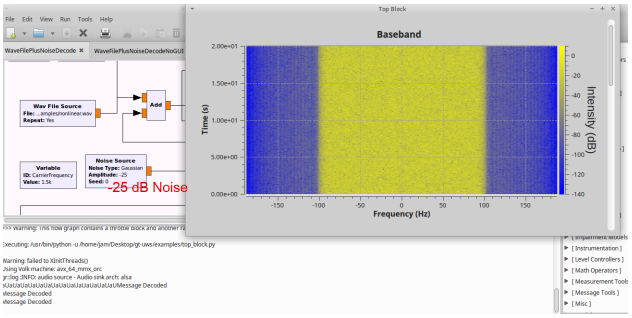

Figure 10. Decoding a two frames with noise

environment, while being subjected to very high noise levels (-25 dB SNR, 2.5 kHz bandwidth reference) and linear or nonlinear frequency shifts. They are capable of handling Doppler due to mobility of the communicating vehicles (both linear and nonlinear). Source code and examples are available online: github.com/michelbarbeau/gr-uwspr.

References

Ahmad, Abdel-Mehsen, Barbeau, Michel, Garcia-Alfaro, Joaquin, Kassem, Jamil, Kranakis, Evangelos, and Porretta, Steven. Doppler effect in the underwater acoustic ultra low frequency band. In *Proceedings of the 9th EAI International Conference on Ad Hoc Networks, Niagara Falls, Canada, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 223*, pp. 3–12. Springer, Cham, 2017.

Ahmad, Abdel-Mehsen, Barbeau, Michel, Garcia-Alfaro, Joaquin, Kassem, Jamil, Kranakis, Evangelos, and Porretta, Steven. Doppler effect in the acoustic ultra low frequency band for wireless underwater networks. *Mobile Networks and Applications*, pp. 1–11, 2018a. https://link.springer.com/article/10.1007/s11036-018-1036-9.

Ahmad, Abdel-Mehsen, Barbeau, Michel, Garcia-Alfaro, Joaquin, Kassem, Jamil, Kranakis, Evangelos, and Porretta, Steven. Low frequency mobile communications in underwater networks. In *17th International Conference on Ad Hoc Networks and Wireless (AdHoc-Now)*, St. Malo, France, 2018b.

Antonelli, Gianluca. Underwater robots. In *Springer Tracts in Advanced Robotics*, volume 96, pp. 987–1008. Springer, 2014.

Barbeau, Michel. Weak signal underwater communications in the ultra low frequency

band. In *Proceedings of the 7th GNU Radio Conference*, San Diego, CA, U.S.A., 2017. https://pubs.gnuradio.org/index.php/grcon/article/view/20/14.

Button, Robert W., Kamp, John, Curtin, Thomas B., and Dryden, James. A survey of missions for unmanned undersea vehicles. RAND National Defense Research Institute, 2009.

Decarpigny, J., Hamonic, B., and Wilson, O. The design of low frequency underwater acoustic projectors: present status and future trends. *IEEE Journal of Oceanic Engineering*, 16(1):107–122, 1991.

Fano, R. A heuristic discussion of probabilistic decoding. *IEEE Transactions on Information Theory*, 9(2):64–74, April 1963.

Freitag, L., Partan, J., Koski, P., and Singh, S. Long range acoustic communications and navigation in the arctic. In *OCEANS 2015 - MTS/IEEE Washington*, pp. 1–5, October 2015.

Hixson, E. A low-frequency underwater sound source for seismic exploration. *The Journal of the Acoustical Society of America*, 126(4):2234–2234, 2009.

Marage, J.P. and Mori, Y. *Marage, J.P., Mori, Y.: Sonar and Underwater Acoustics*. Wiley, 2010.

Otnes, R., Voldhaug, J. E., and Haavik, S. On communication requirements in underwater surveillance networks. In *OCEANS 2008-MTS/IEEE Kobe Techno-Ocean*, pp. 1–7. IEEE, 2008.

Otnes, Roald, Asterjadhi, Alfred, Casari, Paolo, Goetz, Michael, Husøy, Thor, Nissen, Ivor, Rimstad, Knut, Van Walree, Paul, and Zorzi, Michele. *Underwater acoustic networking techniques*. Springer Science & Business Media, 2012.

Proakis, J.G. and Salehi, M. *Digital Communication*. McGrah-Hill Higher Education, fifth edition, 2008.

Stojanovic, M. On the relationship between capacity and distance in an underwater acoustic communication channel. *SIGMOBILE Mob. Comput. Commun. Rev.*, 11(4): 34–43, October 2007. ISSN 1559-1662.

Wikipedia. The WSPR (Weak Signal Propagation Reporter) amateur radio software. https://en.wikipedia.org/wiki/WSPR_(amateur_radio_software), 2018a.

Wikipedia. Underwater locator beacon. https://en.wikipedia.org/wiki/Underwater_locator_beacon, 2018b.