TU Delft ET4394
Wireless Networking
Course Project

LoRa receiver SDR implementation in MATLAB using the RTL-SDR dongle

Guillermo Ortas 4612957
Abdul Aziz Hamad 4621565
Kumar Navneet 4619110

March 6, 2017

1 Introduction and scope

The project aims to develop a software defined radio LoRa receiver implementation in MATLAB, using the RTL-SDR USB dongle (RTL2832U with R820T2 tuner) as a radio receiver. LoRa operates in the ISM band and refers only to the physical layer of the Low-Power Wide-Area Network (LPWAN) standard. LoRa is completed by LoRaWAN, a media access control (MAC) layer protocol. LoRa uses a proprietary chirp spread-spectrum (CSS) modulation owned by Semtech which enables large distance and low bit rate communications, having in mind IoT devices [1].

As discussed later, implementing a general LoRa receiver is subject to the fact that a random sequence is used to whiten the data before being transmitted. This sequence is only known if the receiver is an official device. In our case the only way to get it is to send known data beforehand.

2 Initial research

High level information about this technology is provided by the LoRa Alliance itself and TTN, but information of how it actually works at a lower level is not disclosed due to being a patent pending modulation. Also a lot more information about LoRaWAN can be found than about LoRa.

Luckily, a software engineer from Bastille Research named Matt Knight has lately been working on reverse-engineering the modulation and developing a python open-source for Gnu-Radio implementation of the LoRa physical layer [2]. This implementation is quite well documented and explained and it will be used as a reference throughout this whole project. A lot of information and understanding of the inner working of LoRa was extracted from this work.

These are some general and basic but very important points about the modulation to take into account at all times:

- The modulation can be understood as a MFSK modulation on top of a chirp signal.
- The used spreading factor is the number of bits per symbol and ranges from 6 to 12.
- The bandwidth of the channels can either be 125 kHz, 250 kHz or 500 kHz.
- The chirp rate is $\frac{BW}{2^{SF}}$ and is the inverse of the symbol time.
- The preamble/training sequence consists of 10 consecutive up-chirps (the first two are a special sync word), corresponding to the value zero.
- The Start of Frame Delimiter (SFD) is made up of two and a quarter down-chirps, immediately followed by the data.

3 Implementation

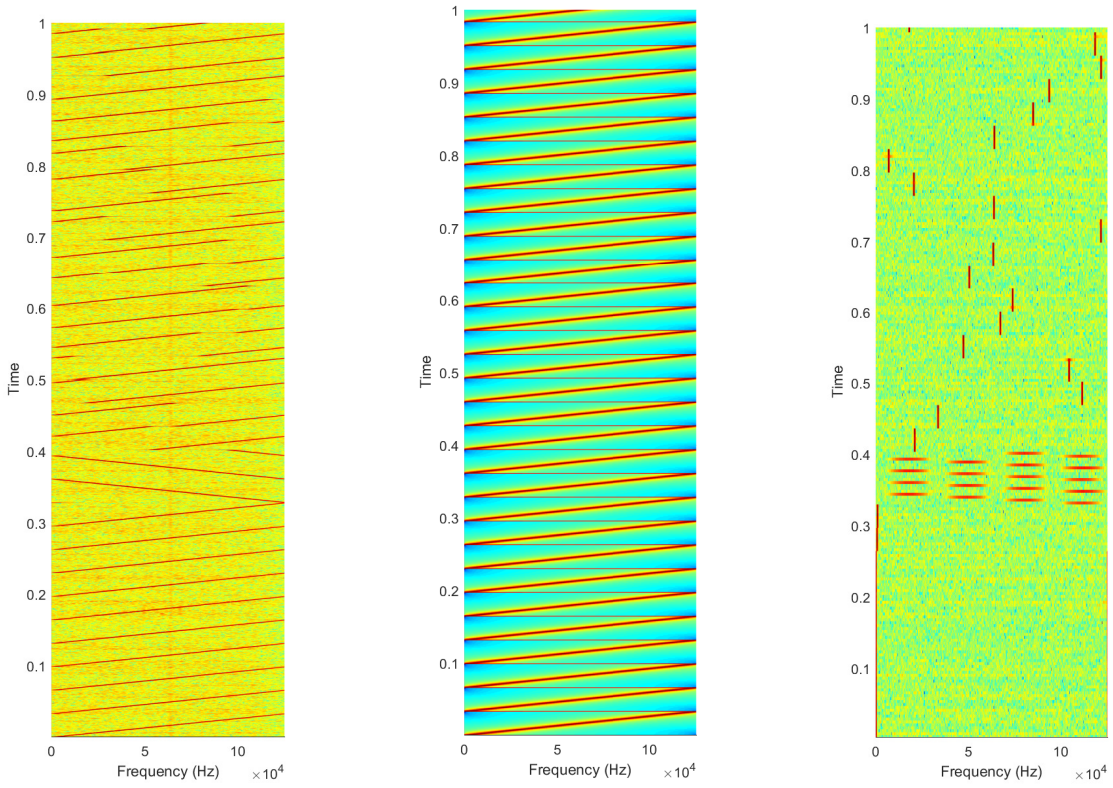
The project is divided into two distinct main blocks: demodulation and decoding. The first block is about translating the raw LoRa RF signal into bits and the second block is about processing those acquired bits to actually get the transmitted data represented by those bits.

3.1 Demodulation block

To successfully demodulate LoRa and extract the transmitted symbols, a processing pipeline was set up. It consists on: channelization and resampling, synchronization, de-chirping, spectrogram conditioning and finally symbol and extraction.

During this project, both real signals captured by ourselves and prerecorded samples generated by Matt Knight (which contain known data) will be used with the objective of making sure everything works with different parameters and conditions of reception. That being said, figures will show one of our own captured signals.

The LoRa signal in figure 1a was recorded at 2.4 Msps, has a bandwidth of 125 kHz and its spreading factor is 12. These parameters were acquired by inspecting the spectrogram and they are needed in the process of demodulation. In the spectrogram we can also very easily identify the 10 up-chirps of the preamble and the 2.25 down-chirps of the SFD, followed by choppy chirps, which contain the data.



(a) Captured LoRa signal. (b) Locally generated up-chirp. (c) De-chirped signal.

Figure 1: Spectrograms of various signals.

3.1.1 Channelization and re-sampling

To be able to work with the signal much more easily and efficiently, it is channelized using a Digital Down Conversion (DDC) technique. In this process the signal is first brought to base-band, then

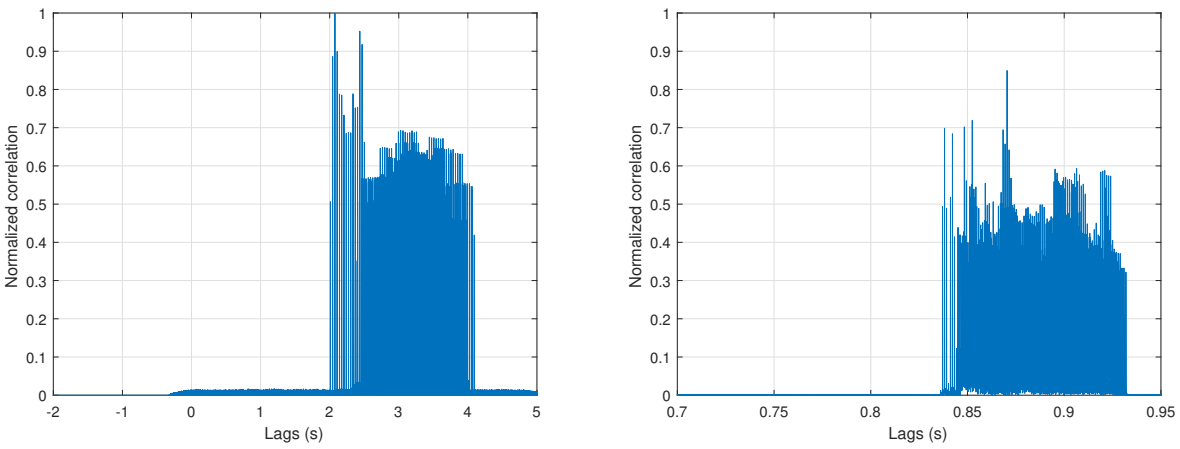
low pass filtered and finally the sampling rate is reduced to match the chirp bandwidth, satisfying the Nyquist-Shannon theorem.

In SDR, a signal is typically captured at a sampling frequency that can range from 1 to 10 MHz. For a LoRa packet of 2 seconds of duration, that is a lot of samples. This process dramatically decreases the computational load required to process the signals since only the necessary samples are left, which are a fraction of the original recorded ones.

3.1.2 Synchronization

When a signal recording is fed to be processed, the exact instant at which the actual signal starts is not known. This is not only important to not process an unnecessary amount of data, but it is also critical for symbol synchronization. If there is a time offset between the signal and the demodulating chirp a symbol's power will leak over to adjacent symbols, which makes it much harder to identify them correctly, as it's quite difficult to tell which of the two powerful spectral components in the same symbol time is the right one.

To achieve this synchronization, the correlation is taken between the input signal and a local up-chirp of the same parameters (the next subsection explains how it is generated). Thanks to the good correlation properties of chirp signals, the start of the preamble is easily identified using a dynamic threshold level, which was heuristically set to a quarter of the maximum correlation value, acting as a normalization factor. Figure 2 shows correlation functions for two different signals.



(a) Correlation of signal 1.

(b) Correlation of signal 2.

Figure 2: Correlation functions of a local up-chirp with incoming signals.

The correlation lag of the first peak above the set threshold will correspond to the sample at which the preamble starts, so the signal is cropped in time according to that index. The correlation function can not only be useful for identifying the start of the signal, but also for figuring out for how long it lasts and the moment it ends. For example in figure 2a the correlation function tells that signal 1 spans from approximately second 2 of the recording to second 4, whereas in 2b it can be seen that signal 2 does so from 0.84s to 0.93s. The higher, more spaced peaks at the beginning of either case correspond to the ten full up-chirps that make up the preamble.

After perfectly locating the start of the frame, 1/4 of a symbol time is added to the start index to account for the 0.25 extra down-chirps at the end of the SFD, which would otherwise ruin the

synchronization in the data section of the signal and make symbol power leak into adjacent ones. This effect is shown figure 3, which compares the unsynchronized data segment in 3a and the fully synchronized data segment in 3b.

Experimental results show that this method has a minimum accuracy of two lag samples in the worst case. Since the sample rate is set to match the bandwidth of the signal, the absolute error depends only on the bandwidth of the signal and the relative error only on the spreading factor. Table 1 shows different LoRa configurations (including the longest and shortest possible symbol lengths) and their corresponding absolute and relative errors.

Table 1: Synchronization errors for various LoRa configurations.

LoRa parameters	Symbol time (ms)	Maximum absolute error (μs)	Maximum relative error (%)
SF=12, BW=125kHz	32.768	16	0.0488
SF=12, BW=500kHz	8.192	4	0.0488
SF= 6, BW=125kHz	0.512	16	3.125
SF= 6, BW=500kHz	0.128	4	3.125

Although 3% may seem more than desirable, in practice results show that almost perfect synchronization is achieved and that symbol power is nicely contained within its corresponding slot as demonstrated in figure 3. This makes symbol extraction much more reliable later on, as it boosts symbol energy and facilitates their identification.

3.1.3 De-chirping

First, a chirp with the adequate chirp rate matching the one of the incoming signal is generated. This is done with the help of the built-in `chirp` Matlab function. The resulting local chirp is a complex up-chirp and it is centered around zero frequency, same as the channelized signal. First, a single chirp is generated with this procedure and then it is repeated and cropped to match the number of samples of the input signal. The final up-chirp is depicted in figure 1b, in the figure

Finally, the incoming signal is multiplied with the complex conjugate of the locally generated chirp (which will be a down-chirp). When two signals of opposite frequencies are multiplied, the result is a baseband signal. This operation undoes the chirping and compresses all the symbol energy (spread over the bandwidth of the signal) into symbol bins of constant frequency, which are much easier to work with. The resulting signal of this operation is equivalent to a MFSK modulation with M being 2^{SF} levels.

The de-chirped signal is shown in figure 1c, in which the spectrogram parameters were adjusted to make the symbols easier to see. The horizontal lines after the preamble correspond to the SFD, which is made out of down-chirps instead of up-chirps. De-chirping the SFD with another down-chirp (not being its complex conjugate) further spreads the energy rather tan compressing it into symbols, but that does not matter because it only acts as a delimeter and does not contain any other useful information.

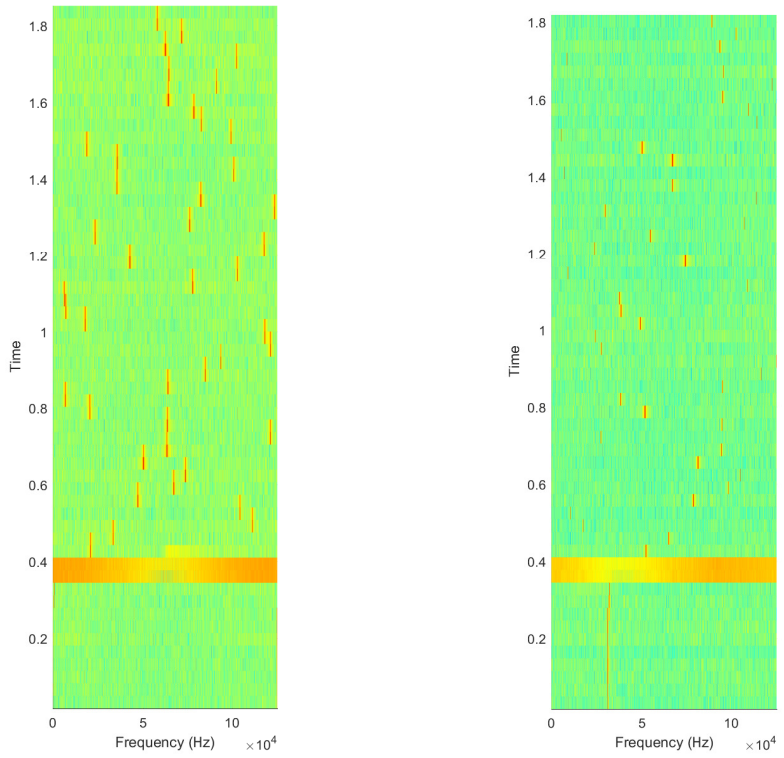
Note that all signals in figure 1 are actually centered around zero hertz and are only shown in the range $[0,\ BW]$ for easy interpretation.

3.1.4 Spectrogram conditioning

A spectrogram works by making use of a short-time Fourier transform (STFT) technique, which takes time windows or segments of the signal of the specified length and computes a fast Fourier transform (FFT) for each of them. Each FFT will in turn give back an specified number of frequency samples, which for efficiency of the implementation is a power of two. Stacking all these FFTs, one can see how the spectral content of a signal changes over time.

Since after de-chirping we were left with a MFSK modulation and the signal was channelized to its bandwidth BW, by matching the FFT lengths to the number of possible symbols that can be coded in BW, we can establish a one-to-one correspondence between possible symbol values and FFT frequency bins. This makes it very easy to extract the transmitted symbols in the next step.

As explained in the initial research, the number of bits per symbol is called the spreading factor SF, and so there are 2^{SF} possible transmitted symbol values. We then need to set the FFT lengths to this value, and symbols will be in the range $[0,\ 2^{SF} - 1]$.



(a) Unsynchronized data segment. (b) Synchronized data segment.

Figure 3: Adjusted spectrograms of de-chirped signals.

After synchronization the signal was set to begin just at the start of a symbol. It is needed to compute exactly one FFT per symbol, so the length of the time windows for computing the spectrogram just needs to match the length of a symbol to have correct results. It is known that a symbol duration is $\frac{2^{SF}}{BW}$ seconds (inverse of the chirp rate), so the amount of samples in one symbol is $\frac{2^{SF}}{BW} F_s$. Furthermore, in the channelization section the signal was resampled to its bandwidth

($F_s' = BW$), meaning that the samples in a symbol are then 2^{SF}. The window lengths and FFT lengths for computing the spectrogram are both 2^{SF}.

With these settings the spectrogram will give exactly one sample per frequency bin per symbol time, the resulting adjusted spectrogram is shown in figure 3b. Everything is now ready for symbol extraction.

3.1.5 Extraction of symbols and bits

Having correctly set up the spectrogram to return one sample per symbol per frequency bin, it is just a matter of finding the index of the strongest spectral component for each symbol time, which corresponds to the transmitted symbol at that instant.

There is one more detail to keep in mind: in the synchronization process the incoming signal and the demodulating chirp were perfectly set to start at the same time, and then a time offset of a quarter symbol time was introduced to align the chirps in the data section. As we know, a phase offset ($\pi/4$ in this case) in the time domain corresponds to an offset in the frequency of that signal. As the symbols are encoded in the frequency bin they occupy in the spectrum, what this means is that there is an offset corresponding to 0.25 times the maximum possible symbol value. In other words, there is a positive symbol value offset of $\frac{1}{4}2^{SF} = 2^{SF-2}$. As the symbols start at zero and not at one, we finally need to subtract one to the result. Figure 3b exemplifies this phenomenon, as the preamble if shifted one quarter of the way to the right (a $2^{12-2} = 1024$ shift in symbol value).

To remove this offset it is just needed to subtract this number of symbols to all identified values and then perform a 2^{SF} modulo operation to make sure all of them remain in the correct range.

It is known that LoRa encodes all the symbols in reference to the preamble, which always carries the value zero. So as an extra step it can be checked that $2^{SF-2} - 1$ is indeed the identified symbol value of the preamble before compensating the offset. This strategy could also be used to normalize the symbols instead.

Finally, to get the raw bits that have been transmitted, the resulting symbol values are just translated into base 2.

3.1.6 Final remarks

WORK HERE: explain more

Although this process may seem straightforward and even somewhat intuitive it was definitely not as easy to implement. A deep understanding of how these signals behave and their interaction is required, also lots of tests were needed to produce this seemingly simple implementation.

It is worth mentioning that different approached in the de-chirping process were considered and tested, for example..... In that case, after de-chirping, the symbols would have constant frequency but would be spread and split over a bandwidth of $2BW$. This means that some segments of symbols were located in the range $[0,\ BW]$ and the rest in $[BW,\ 2BW]$.

3.2 Decoding block

Explain exactly WHAT we're going to do and elaborate on HOW specifically we're going to do so. Go through the planned steps in as much detail as possible.

References

[1] Semtech, "Lora™ modulation basics," *AN1200.22*, [Online]. Available: http://www.semtech.com/images/datasheet/an1200.22.pdf.

[2] M. Knight, "Rversing lora - exploring next generation wireless," *GNU Radio Conference 2016*, [Online]. Available: https://drive.google.com/file/d/0B6ccrJyAZaq3VmItaThiVEtBeVU/view.

[3] Semtech, "Lora modem - designer's guide," *AN1200.13*, [Online]. Available: https://www.semtech.com/images/datasheet/LoraDesignGuide_STD.pdf.

[4] rpp0, "Gnuradio blocks for receiving lora modulated radio messages using sdr," *GitHub*, [Online]. Available: https://github.com/rpp0/gr-lora.

[5] BastilleResearch, "Gnu radio oot module implementing the lora phy," *GitHub*, [Online]. Available: https://github.com/BastilleResearch/gr-lora.