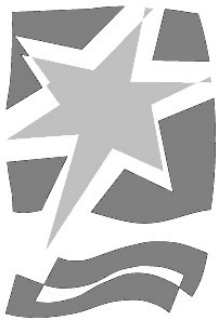
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITÈCNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



ESCOLA POLITÈCNICA
SUPERIOR DE GANDIA

"Implementación de un receptor banda base de TETRA con SDR en Matlab "

TRABAJO FINAL DE GRADO

Autor/a:
Garavito Vasquez, Fernando

Tutor/a:
Almenar Terré, Vicenç

Cotutor/a externo:
ELDERS-BOLL, HARALD

GANDIA, 2023

Implementation of an SDR TETRA Baseband Receiver in MATLAB®

Bachelor thesis for the award of
Bachelor of Science in the study program Electrical Engineering
at the Faculty of Information, Media and Electrical Engineering
of the Cologne University of Technology

submitted by:	Fernando Garavito Vásquez
Matriculation No.:	11138956
E-mail:	fernando.garavito_vasquez@smail.th-koeln.de, fergava1@upv.es

submitted to:	Prof. Dr.-Ing. Harald Elders-Boll (TH Köln)
Second supervisor:	Prof. Dr.-Ing. Vicenc Almenar Terre (UPV)

Cologne, 27.07.2022

Technology
Arts Sciences
TH Köln

Abstract

This project studies the requirements for capturing a TETRA radio signal with an RTL-SDR device and the processing of the captured data with the aid of MATLAB®.

For this purpose, a cheap RTL-SDR in combination with some blocks coded in MATLAB® is used to prove the ease of access we have today to powerful tools and that was in the past out of our reach.

This project includes a large application of the knowledge acquired during the degree of Telecommunication Engineering, especially the signal processing subject and might be useful for a future reference for newer students.

It is presented by stages, from the basic technical description of the TETRA signal and going through every step of the signal processing, including a brief review of the theory for every programmed block.

Keywords: TETRA, SDR, DSP, MATLAB, Physical layer

TABLE OF CONTENTS

List of figures

Introduction

With the evolution of the electronic devices, faster processing speed and bigger memory size have helped to move from the hardware-based receivers to the software defined radios. In the past all stages in the communications field where hardware based, from the local oscillator to the demodulator, but in the present day with the powerful processors we can implement some stages as a software, allowing to save a lot of physical resources.

With this Bachelor thesis, I wanted to explore the possibilities of this trending technology. As a comparison, a cheap TETRA receiver cost around 200€, and the RTL-SDR used in this project cost around 15€, even though the receiver is not fully functional as some stages are missing, but it can be still completed by adding the missing stages to the point of having a fully functional receiver.

This project also pretends to review the theory behind the implemented blocks and tried to explain it in a clear way, so anybody interested in this project can understand it without requiring an advanced knowledge, as some chapters from the books aren’t easy to understand at first.

1. TETRA MAIN FEATURES

The TETRA system is widely used in the world by public institutions and emergency services. The frequency assignation is divided into two main groups: Emergency and Public services. The assigned frequencies are shown in the table below:

EMERGENCY SERVICES

	Band 1 (MHz)	Band 2 (MHz)
1	380-383	390-393
2	383-385	393-395

PUBLIC SERVICES

	Band 1 (MHz)	Band 2 (MHz)
1	410-420	420-430
2	870-876	915-921
3	450-460	460-470
4	385-390	395-399,9

The carrier spacing is 25 kHz with a traffic channel bandwidth of 25 kHz/4 = 6,25 kHz for Voice + Data (trunked) and DMO (direct mode operation) or 25 kHz for PDO (packet data optimized).

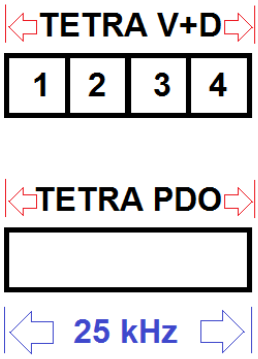


Figure 1.1 TETRA carrier spacing

TETRA uses TDMA for multiple access and FDD as multiplexing system with a separation for the UL and DL of 10 MHz in the case of VHF and 45 MHz for UHF. The data is modulated on a π/4 D-QPSK modulation scheme with a modulation rate of 36 kb/s, the pulse is based on a root raised cosine filter with a roll-off of 0,35.

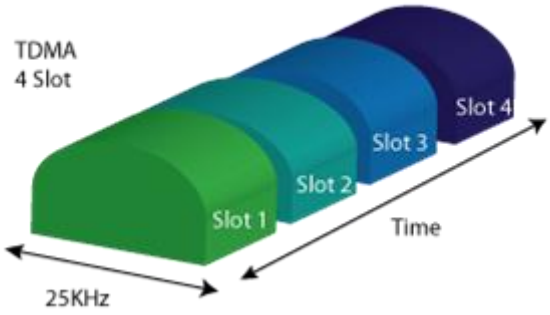


Figure 1.2 TETRA TDMA slots

The π/4 D-QPSK modulation uses two identical constellations, rotated by 45° (π/4 in radians). The transitions of the symbols in this modulation scheme never pass through the origin, lowering the dynamical range of fluctuations in the signal.

In the Figure 1.3 these concepts are represented, on the right the two constellations are shown with different colours, separated by 45°. On the left the possible symbols transitions are drawn, as said before it never goes through the origin.

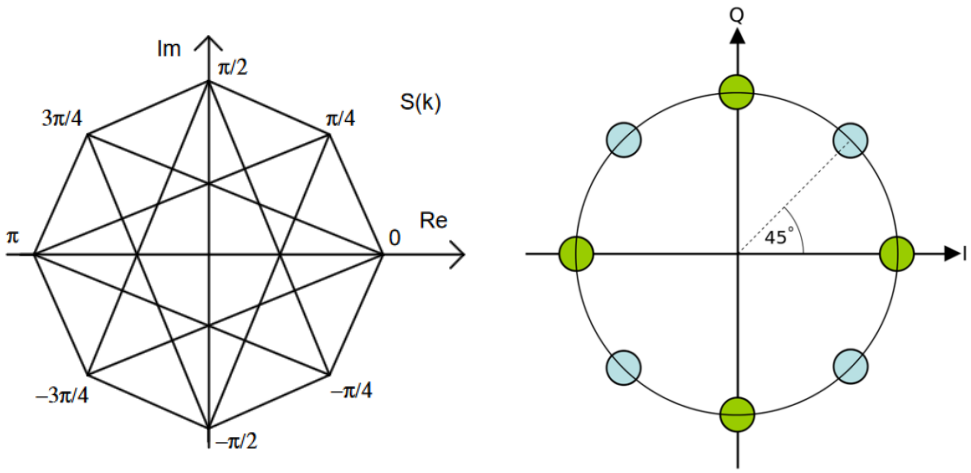


Figure 1.3 π/4 D-QPSK symbols transitions and constellation

1.1 TETRA BURST AND FRAMES

The TETRA system is based on time slots, 4 time slots form a frame, and many frames can form a bigger structure called multiframe and hyperframe, as shown in the figure below.

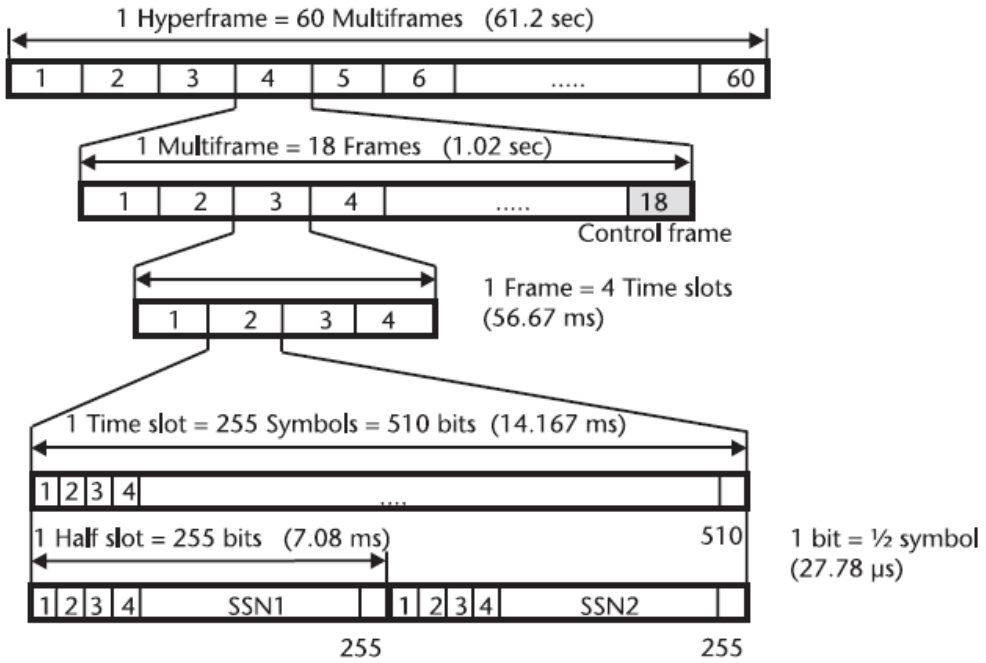


Figure 1.4 TETRA frame structure

One time slot is known as burst, and it contains 255 modulation symbols. The are 8 types of bursts, but since the aim of the project is to create a receiver, only the downlink frames will be taken in count. The fields and contents of the bursts won't be described, as only the training sequences will be used to test the correct demodulation of the symbols.

In the case of downlink bursts, we have:

1. Normal continuous downlink burst

BIT NUMBER	***CONTENT***	
1-12	*normal training sequence 3*	*0,0,0,1,1,0,1,0,1,1,0,1*
13-14	*phase adjustment bits*	
15-230	*scrambled block 1 bits*	
231-244	*scrambled broadcast bits*	
245-266	*normal training sequence*	*1,1,0,1,0,0,0,0,1,1,1,0,1,0,0,1,1,1,0,1,0,0 **OR** 0,1,1,1, 1,0,1,0,0,1,0,0,0,0,1,1,0,1,1,1,1,0*
267-282	*scrambled broadcast bits*	
283-498	*scrambled block 2 bits*	
499-500	*phase adjustment bits*	
501-510	*normal training sequence 3*	*1,0,1,1,0,1,1,1,0,0*

2. Synchronization continuous downlink burst

BIT NUMBER	***CONTENT***	
1-12	*normal training sequence 3*	*0,0,0,1,1,0,1,0,1,1,0,1*
13-14	*phase adjustment bits*	
15-94	*frequency correction*	*Bits 1:8 = 1 Bits 9:72 = 0 Bits 73:80 = 1*
95-214	*scrambled synchronization block 1 bits*	
215-252	*synchronization training sequence*	*1,1,0,0,0,0,0,1,1,0,0,1,1,1,0,0,1,1,1,0,1,0,0,1,1,1,0,0,0,0, 0,1,1,0,0,1,1,1*
267-282	*scrambled broadcast bits*	
283-498	*scrambled block 2 bits*	
499-500	*phase adjustment bits*	
501-510	*normal training sequence 3*	*1,0,1,1,0,1,1,1,0,0*

3. Normal discontinuous downlink burst

BIT NUMBER	***CONTENT***	
1-2	*normal training sequence 3*	*0,1*
3-4	*phase adjustment bits*	
5-220	*scrambled block 1 bits*	
221-234	*scrambled broadcast bits*	
235-256	*normal training sequence*	*1,1,0,1,0,0,0,0,1,1,1,0,1,0,0,1,1,1,0,1,0,0 **OR** 0,1,1,1, 1,0,1,0,0,1,0,0,0,0,1,1,0,1,1,1,1,0*
257-272	*scrambled broadcast bits*	
273-488	*scrambled block 2 bits*	
489-490	*phase adjustment bits*	
491-492	*normal training sequence 3*	*1,0*

4. Synchronization discontinuous downlink burst

BIT NUMBER	***CONTENT***	
1-2	*normal training sequence 3*	*0,1*
3-4	*phase adjustment bits*	
5-84	*frequency correction*	*Bits 1:8 = 1 Bits 9:72 = 0 Bits 73:80 = 1*

85-204	*scrambled synchronization block 1 bits*	
205-242	*synchronization training sequence*	*1,1,0,0,0,0,0,1,1,0,0,1,1,1,0,0,1,1,1,0,1,0,0,1,1,1,0,0,0,0, 0,1,1,0,0,1,1,1*
243-272	*scrambled broadcast bits*	
273-488	*scrambled block 2 bits*	
489-490	*phase adjustment bits*	
491-492	*normal training sequence 3*	*1,0*

In the Normal continuous and discontinuous downlink burst exists 2 different 22 bits long training sequence, these are used to indicate the presence of one or two logical channel.

1.2 FINDING A TETRA SIGNAL

To find a TETRA signal we need to scan over the band frequencies defined in the standard and look out for a spectrum with 25 kHz bandwidth. For this purpose, we use a free software called SDR# from AirSpy, we set the frequency at 390Mhz (Band 2) where a signal centred at 390,2875 MHz is found:

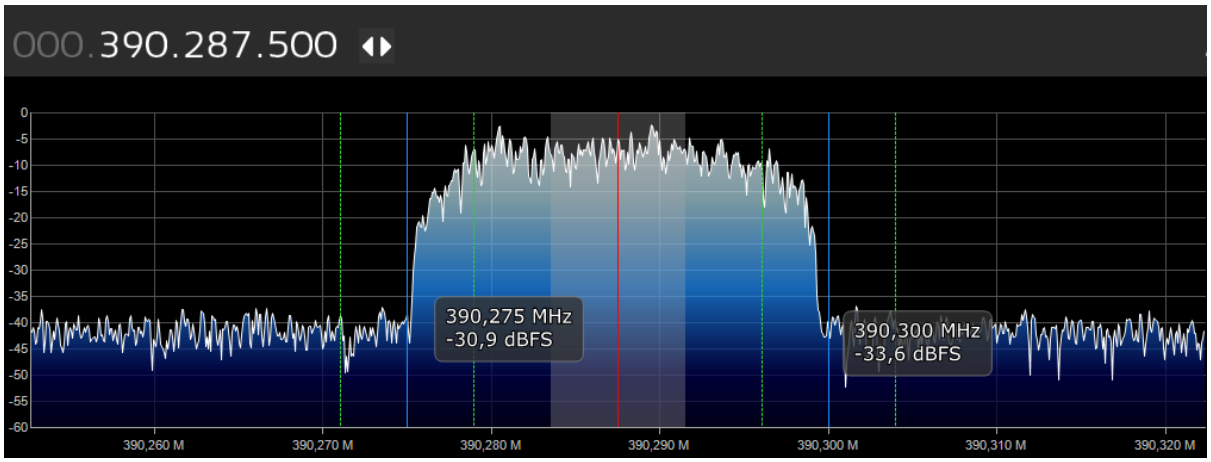


Figure 1.5 Spectrum of a TETRA signal found in SDR#

Now with this value, the SDR object can be configured in MATLAB® as the center frequency parameter.

2. NooElec R820T SDR

This device is based on the chip R820T from Rafael Micro, a low-power tuner for digital television, including the standards DVB-T, ATSC, DMB-T, ISDB-T. It includes a high precision 0.5 PPM TCXO crystal.

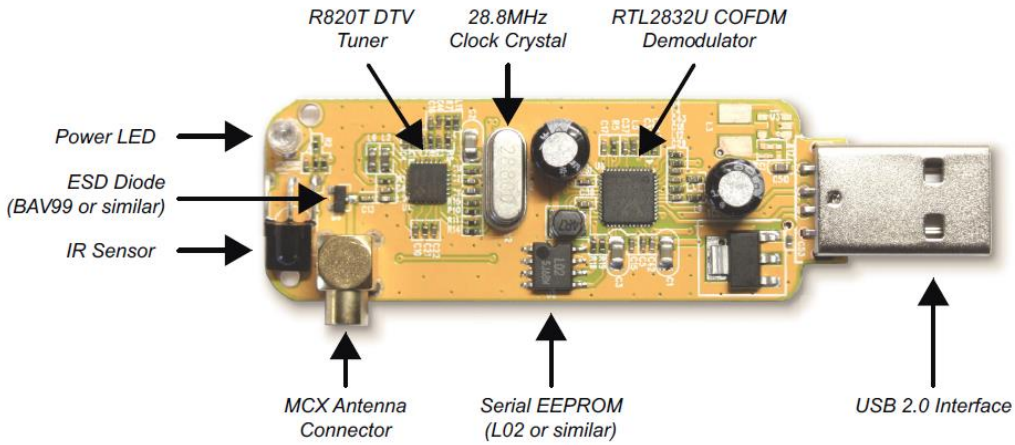


Figure 2.1 Internal components of the RTL-SDR device

It can work in a range of frequencies from 25 MHz to 1750 MHz, this allows us to work with signals in VHF and UHF. The R280T is used for the analog RF to IF process, and then this data is processed by the RTL2832U.

The RTL2832U chip it's a COFDM demodulator, which downconverts the tuned IF signal to baseband and reduces the sampling rate. In this component the tuned signal is sampled at 28.8 MHz and then downsampled by a tunable value. According to the manufacturer the maximum sampling frequency is 3.2 MHz, but it is recommended to use a value under 2.8 MHz to avoid samples dropping, in our case the selected value is 2.4 MHz which is equivalent to decimate by 12. This value complies with the suggestion of the manufacturer and is also a multiple integer of the clock, which is computationally efficient.

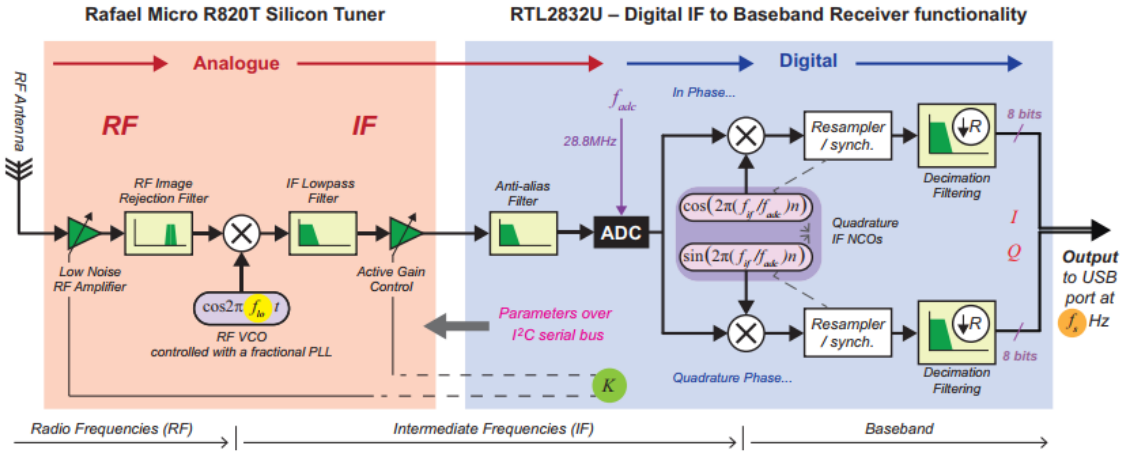


Figure 2.2 R820T SDR signal processing blocks

In the Figure 2.2 can be seen how data is processed into the two branches I and Q (in-phase and quadrature), as result the returned data is in the format $\mathbb{R}+i$ (real for I and imaginary for Q).

As the signal delivered to the computer is already in baseband, we can directly start working with the data without the frequency translation process, saving a lot of computer resources and time.

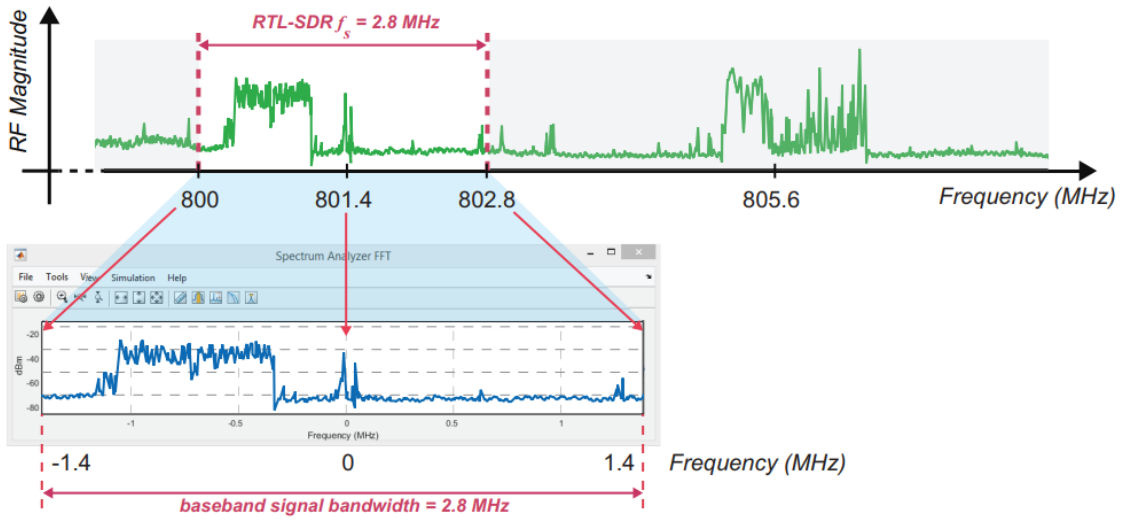


Figure 2.3 Example of Baseband signal delivered by the RTL2832U

2.1 MATLAB® and the RTL-SDR

In the Figure 2.4 the previously explained process is shown as a flow through different blocks, until data in baseband is delivered to MATLAB® for the digital signal processing.

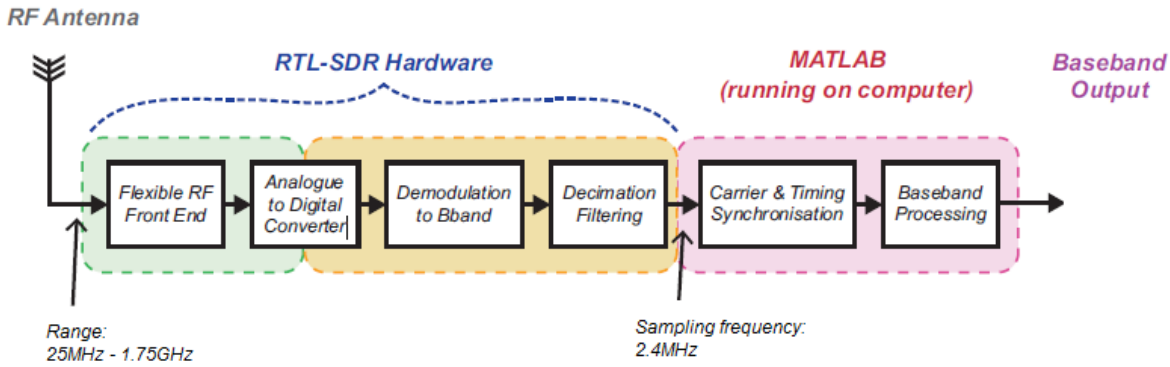


Figure 2.4 Processing blocks of the SDR

In order to capture data with the SDR dongle, we need to install in MATLAB® the corresponding toolbox which can be obtained directly from MATLAB®.

After installation, a new object called `comm.SDRRTLReceiver` will be available, so we only need to create a variable to assign the object and configure the desired parameters.

The code to create the object and set the parameters in MATLAB® is shown below, where `freq` is the center frequency of the TETRA signal, `fs` is the sampling frequency

(as said before we choose 2.4 MHz), `spf` is the number of samples per frame and the output data type is `'double'` for more precision.

```
%% SDR object
rxsdr=comm.SDRRTLReceiver('0');
rxsdr.CenterFrequency=freq;     %center frequency
rxsdr.SampleRate=fs;            %sampling rate
rxsdr.SamplesPerFrame=spf;      %number of samples for 1 frame
rxsdr.OutputDataType='double';
```

There are some more tunable options, but only those of interest are configured, for example, the gain can be chosen but the default option is automatic gain control, and it works fine.

Once the object is created, we can call it with the `step` function assigning a variable to store the data returned by the object. In this case the object is called `rxsdr` and the data will be stored in `rcv`.

```
rcv = step(rxsdr);
```

As the process of capturing data is frame based, every time the object is called, a frame with a length equal to `spf` is then temporary stored in `rcv`.
The value of `spf` is defined as 136000 so the frame time of the SDR will be approximately 1 TDMA frame of TETRA (56.57 ms), as the relation between the samples per frame and frame duration is:

$$Frame\ duration = \frac{samples\ per\ frame(spf)}{sampling\ frequence(fs)} = \frac{136000}{2400000} = 0.0567$$

This frame is then continuously processed in MATLAB® to adapt and correct signal as explained through the next chapters.

2.2 MATLAB® SIMULATION

With the previous configuration, we should receive a frame with a bandwidth of 2.4 MHz, where we can find a TETRA signal located in the center.

A MATLAB® function called periodogram will be useful for this purpose, it is used to graph the power spectral density (PSD). It could be also possible with the object `dsp.SpectrumAnalyzer`, but it is more power consuming.

The required parameters are the variable where data is stored, sampling frequency and type of window and length. In this case we use a Hamming window, and the sampling frequency is the same as the one set for the SDR object.

The command required would be then:

```
periodogram(rcv,hamming(length(rcv)),[],2400000,'centered')
```

And then we obtain the plot of the PSD as shown in the Figure 2.5. The two peaks are TETRA signals.

This signal will be passed through the multi-rate processing stage in order to get the 25 kHz bandwidth signal.

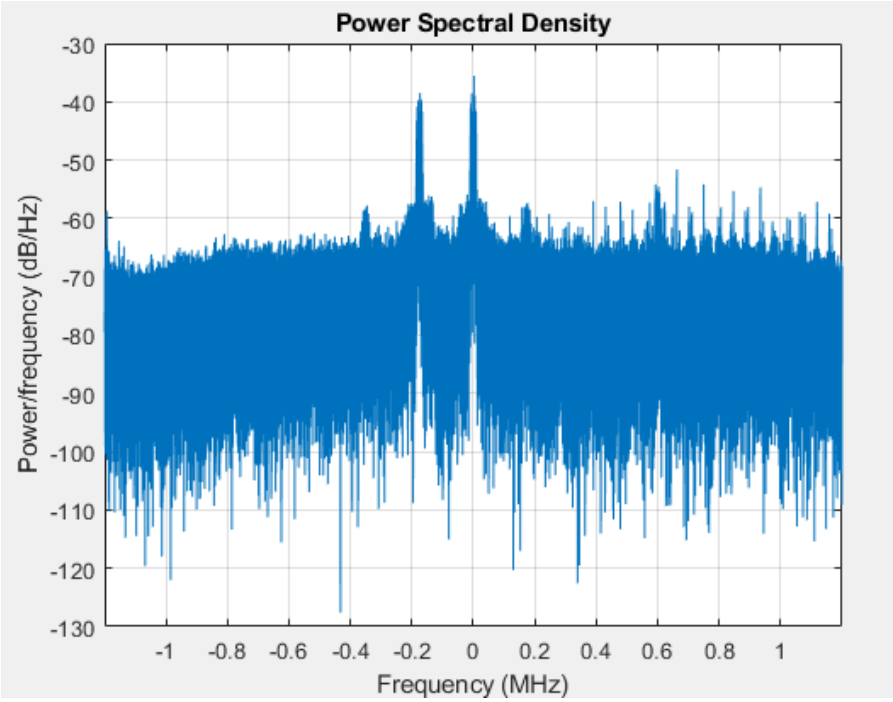


Figure 2.5 PSD plot in MATLAB®

3. MULTIRATE PROCESSING

The signal delivered to MATLAB® is sampled at 2.4 MHz, and the desired signal rate according to the standard (ETSI EN 300 396-2) definition is 18 kHz, this means it is necessary to resample the signal, and this can be achieved with a multirate processing.

It could be also possible to select a lower sampling frequency and avoid some of the filter stages, but it is interesting to show how to work with multirate systems.

The signal will be resampled by the value 3/100, it means the signal is upsampled by 3 and downsampled by 100. In the Figure 3.1 the values of interpolation/decimation are presented, as well the filter used in every stage. The output frequency of every stage is shown in the upper side. These values are selected so the computational resources won't be high demanding.

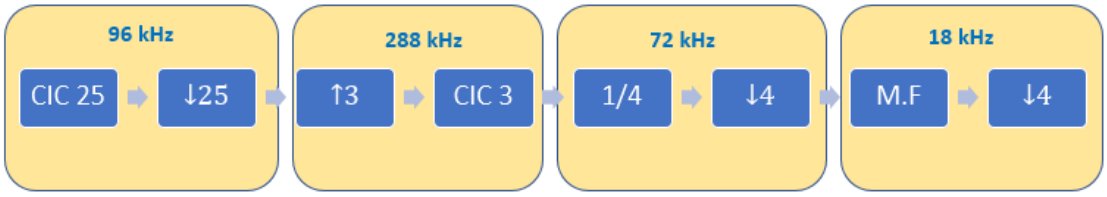


Figure 3.1 Stages of the multirate processing

3.1 CIC FILTERS

Cascade integrator-comb filters are a class of narrowband FIR filters consisting of one or more integrators and comb filters. The operations of a CIC filter include addition and subtraction, compared with traditional FIR filters which also include multiplication. This is useful to save computational resources, and for this reason they are preferred for rate changes larger than 10. The frequency response of CIC filters is sinc(x) like, which has no linear phase, and for this reason a lowpass filter is required before or after, to compensate the non-flat passband of the CIC.

3.1.1 STRUCTURE OF A CIC FILTER

As said before, CIC filters consist of integrators and comb filters, to analyze how it works let's take as example a first order CIC:

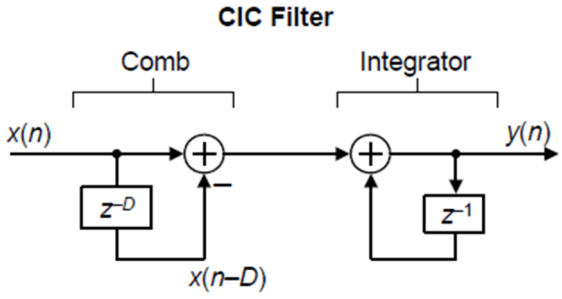


Figure 3.2 Cascade structure of a 1st order CIC

The comb section adds a delayed version of the signal to itself, causing constructive and destructive interference, D is the differential delay. The graph of frequency response is similar to a comb, and that is why it is called like this. In the Figure 3.3 this behavior can be observed. The integrator part of the CIC filter is just an accumulator.

The time domain difference equation of the example is:

$$y(n) = x(n) - x(n-D) + y(n-1)$$

and the z-domain transfer function is:

$$H_{CIC}(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-D}}{1 - z^{-1}}$$

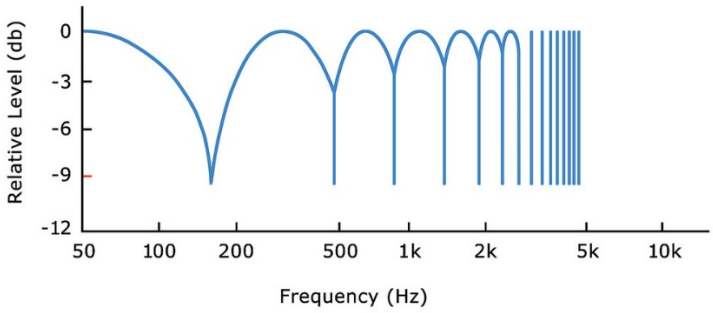


Figure 3.3 Frequency response of a comb filter

The simplest CIC is the all-ones FIR filter ("boxcar"), which is defined as:

$$h(n) = \begin{cases} 1 & 0 \le n < D \\ 0 & otherwise \end{cases}$$

When the CIC is used for decimation, the integrator section comes first, followed by the comb. In the case of interpolation, the structure is reversed. As an example, in the next figure is presented the structure for a single stage CIC.

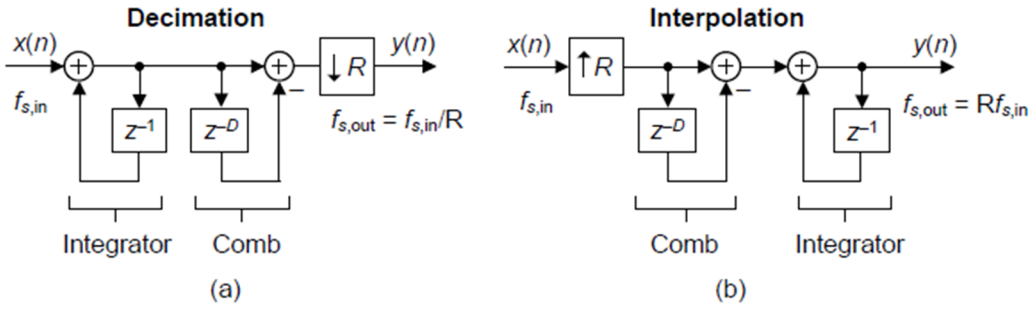


Figure 3.4 Structure of CIC filters for decimation and interpolation

3.1.2 CIC FILTER ORDER

As the notches of the CIC filter fall directly in the center of the frequency images, some aliasing/images will appear. To understand this, it is easier to see the behavior of the spectrum after decimation/interpolation operation.

In the case of decimation, the aliasing occurs due to the spectral folding effect, where all residues from filtering will overlap with the desired signal.

In the Figure 3.6, the desired signal of bandwidth B is filtered before decimation, it can be seen in a) that the images are not completely removed, and then after the decimation they will be overlapping as shown in b).

The case of interpolation is similar, but in this case the result is the generation of some undesired images as in the Figure 3.7, where images appeared at the null centers, that is at every multiple integers of $\frac{f_s}{L}$.

To improve the attenuation of a CIC filter, it is common to connect in cascade many CIC filters, where the order M will be the number of stages connected.

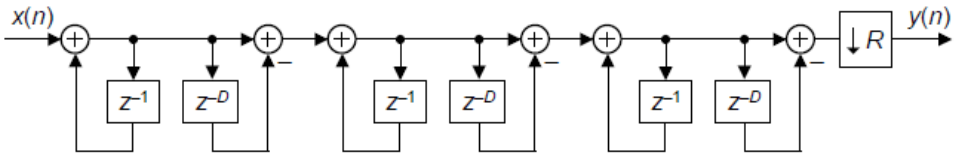


Figure 3.5 CIC of order M=3

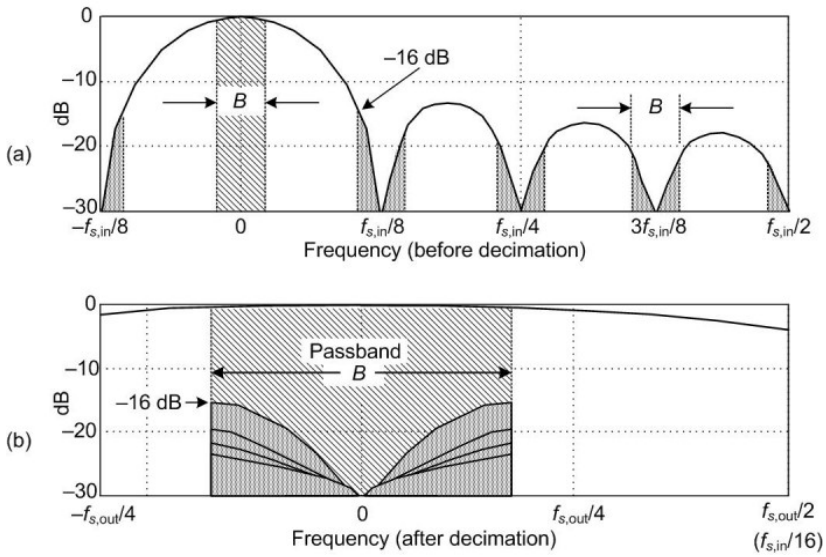


Figure 3.6 CIC aliasing

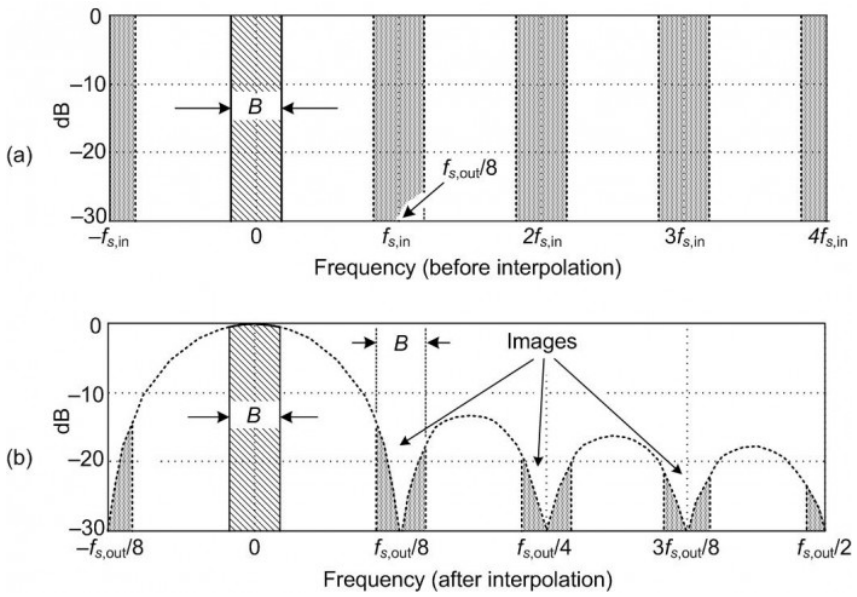


Figure 3.7 Spectral images after interpolation

Increasing the order of the CIC filter, will increase the attenuation. In the practice a second order CIC should be good enough. The Figure 3.8 is the magnitude response of the CIC filter used for decimation by 25 for the TETRA signal. At $\frac{f_s}{25}$ where the first image is located, the attenuation is between -16 and -19 dB, while the Figure 3.9 is the plot of the second order CIC, where attenuation is between -33 and -37 dB.

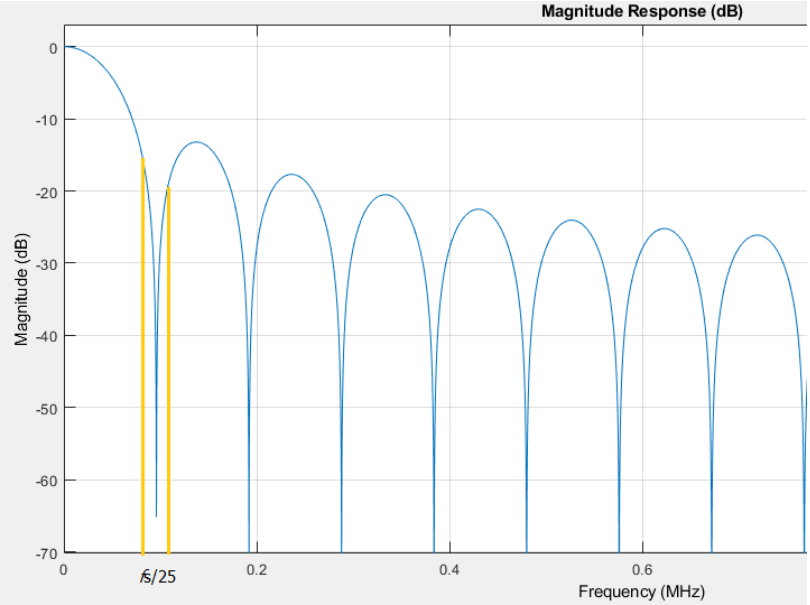


Figure 3.8 First order CIC for decimation

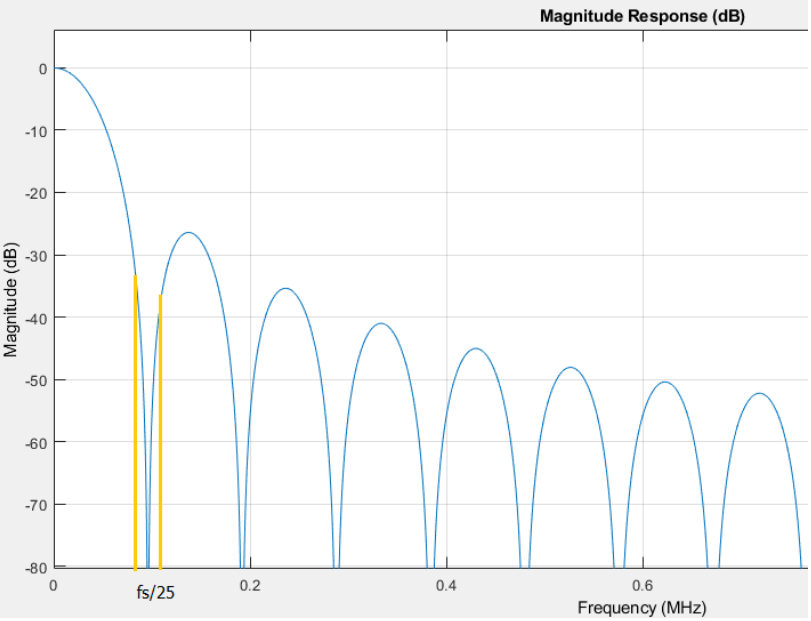


Figure 3.9 Second order CIC for decimation

When designing the CIC filters, it is important to take in count the effects of interpolation and decimation in the sense of gain. To create a CIC filter in MATLAB® is not complicated, first we create an array filled with ones, and the length equal to the desired factor, and then depending on the case we adjust the gain according. In the case of downsampling, the gain must remain in 1, and for interpolation the gain must be set equal to the interpolation factor L.

For example, in the case of the CIC filter for upsampling by 3, the required commands in MATLAB® are:

```
%% CIC filter for upsampling by 3
CIC_up_3_o1 = ones(3,1);                                %%first order CIC
Gain_CIC_up= sum(CIC_up_3_o1);                          %%Gain estimation
CIC_up_3_o1 = CIC_up_3_o1/ Gain_CIC_up;                 %%normalize gain to 1
CIC_up_3_o2 = conv(CIC_up_3_o1,CIC_up_3_o1);            %%second order CIC
CIC_up_3_o2= Gain_CIC_up*CIC_up_3_o2;                   %%set gain to 3
```

3.2 L-TH BAND FILTERS

This type of filters has some important characteristics when they are correctly designed. The first is that every L-th coefficient equals to zero, avoiding the need to multiply them.

The second characteristic is that they are implemented as polyphase filters, and because of the symmetry, the operations can be reduced to the half in some branches.

For this project a ¼-th band filter is used in the downsample stage prior to the matched filter, but before it can be implemented in MATLAB®, some values must be calculated.

The first value is the passband frequency, which can be calculated from:

$$f_p = \frac{1+\alpha}{2*M}$$

Where α is the roll-off factor, in this case 0.35, and M is the number of samples per symbol in this stage, that is 16 samples per symbol. Then we have:

$$f_p = \frac{1+0.35}{2*16} = 0.0422$$

The next value is the elimination band, it can be obtained from the expression:

$$f_e = \frac{1}{L} - f_p$$

Here $\frac{1}{L}$ is the desired L-th band, for this case ¼, and the calculated value is:

$$f_e = 0.25 - 0.0422 = 0.2078$$

Another required value is the attenuation, for 60 dB we have:

$$\delta_e = 10^{\frac{60}{-20}} = 0.001$$

With these values we can use the `firpmord` function, which is a Parks-McClellan optimal equiripple FIR order estimator. This function returns the values required to create the filter through the function `firpm`.

The function will look like this in MATLAB®:

```
%% 1/4 band nyquist filter
[N,Fc,Mc,We] = firpmord([0.0422 0.2078],[1 0],[0.001 0.001],1);
H4bd = firpm(N+2, Fc, Mc, We);
```

N is the order of the filter, Fc contains the band edges frequencies, Mc are the frequency band magnitudes and We the weights. It is important to adjust the order N so it will be even and multiple of the L-th band, otherwise we cannot take profit of the folding

characteristic. The function `firpmord` returns a value of N=18, so the next value fitting the requirements is 20, then N+2 is used.

In the Figure 3.10 the impulse response of the filter is drawn, from this figure we can see the zeros every 4-th sample and the symmetry.

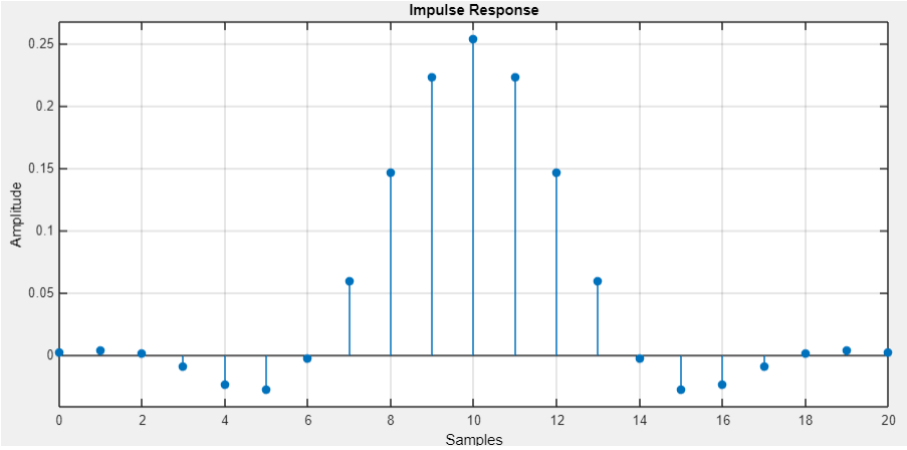


Figure 3.10 Impulse response of the ¼-th band filter

As a polyphase filter the coefficients can be arranged in 4 branches, where one of them will contain the central coefficient (yellow) and the zeros (turquoise). In this branch only one multiplication is necessary.

In the first branch (green) the coefficients are symmetric, therefore only 3 multiplications are necessary.

h_0	h_4	h_8	h_{12}	h_{16}	h_{20}
h_1	h_5	h_9	h_{13}	h_{17}	
h_2	h_6	h_{10}	h_{14}	h_{18}	
h_3	h_7	h_{11}	h_{15}	h_{19}	

With the use of this type of filter we need to perform 14 operations instead of 20, so we have saved some resources. In filters with higher order, resource saving is more evident.

To check out if the filter frequencies are correctly calculated, we can plot the magnitude response of the filter (in MATLAB® the normalized frequency is between 0 and 1):

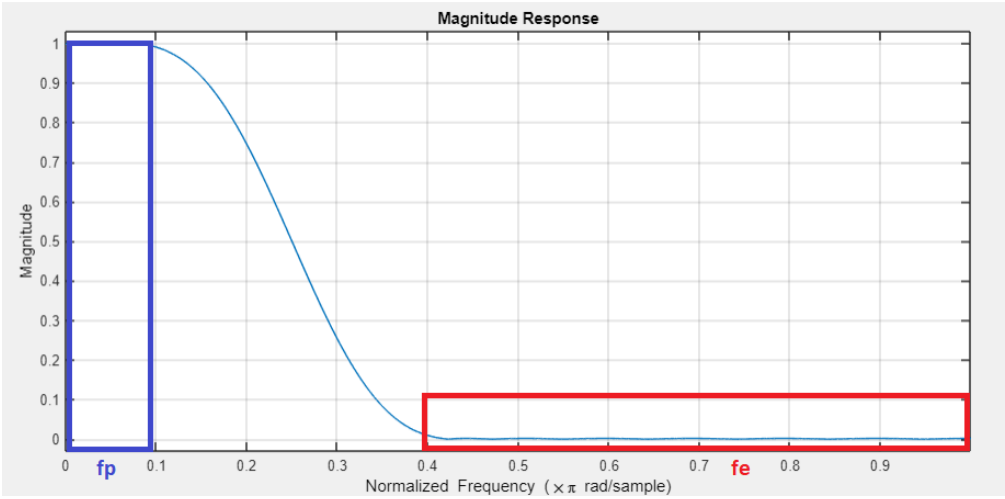


Figure 3.11 Magnitude response of the 1/4-th band filter

3.3 MATCHED FILTER

The last filtered stage is the matched filter, the purpose of this filter is to maximize the Signal to Noise Ratio (SNR). It works by convolving the noisy unknown signal with the matched filter. In the transmitter a pulse shaping filter is used, this makes the signal suitable for transmission by limiting its bandwidth and reduces the Inter Symbol Interference (ISI).

In the case of TETRA signals, a Root Raised Cosine (RRC) filter is used both in transmission and reception. The most important value to take in count is the roll-off factor, sometimes called α and sometimes β depending on the author. As explained before, the received signal is convolved with the matched filter, and since the transmission filter is as well a RRC filter, the convolution will produce a Raised Cosine pulse.

The impulse response of a RRC filter is given by the equation:

$$h(t) = \begin{cases} \frac{1}{\sqrt{T_s}}\left(1-\beta+4\frac{\beta}{\pi}\right), & t=0 \\ \frac{\beta}{\sqrt{2T_s}}\left[\left(1+\frac{2}{\pi}\right)\sin\left(\frac{\pi}{4\beta}\right)+\left(1-\frac{2}{\pi}\right)\cos\left(\frac{\pi}{4\beta}\right)\right], & t=\pm\frac{T_s}{4\beta} \\ \frac{1}{\sqrt{T_s}}\dfrac{\sin\left[\pi\frac{t}{T_s}(1-\beta)\right]+4\beta\frac{t}{T_s}\cos\left[\pi\frac{t}{T_s}(1+\beta)\right]}{\pi\frac{t}{T_s}\left[1-\left(4\beta\frac{t}{T_s}\right)^2\right]}, & \text{otherwise} \end{cases}$$

And the plot of the impulse response of the filter with different roll-off values:

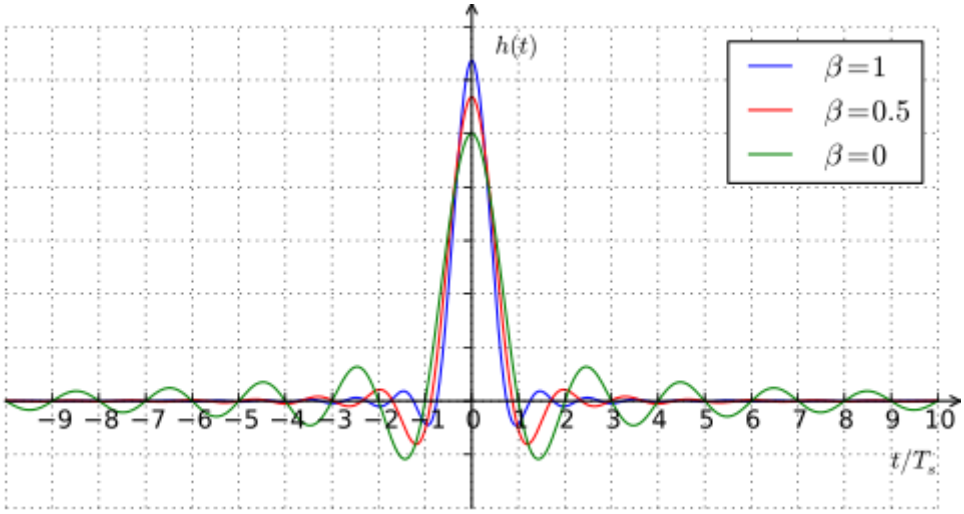


Figure 3.12 Impulse response of the RRC filter

It can be seen in the Figure 3.12 that the impulse response is not zero at the intervals of $\pm T_s$, except for the case of β=0, but after the convolution with the matched filter it will have a zero response in these intervals, as the product is a Raised Cosine shape.

The characteristic of ISI minimization can be seen by plotting consecutive raised-cosine impulses, where at every integer multiple of T, the impulse response of the previous symbol is zero, allowing to transmit the symbol without interferences.

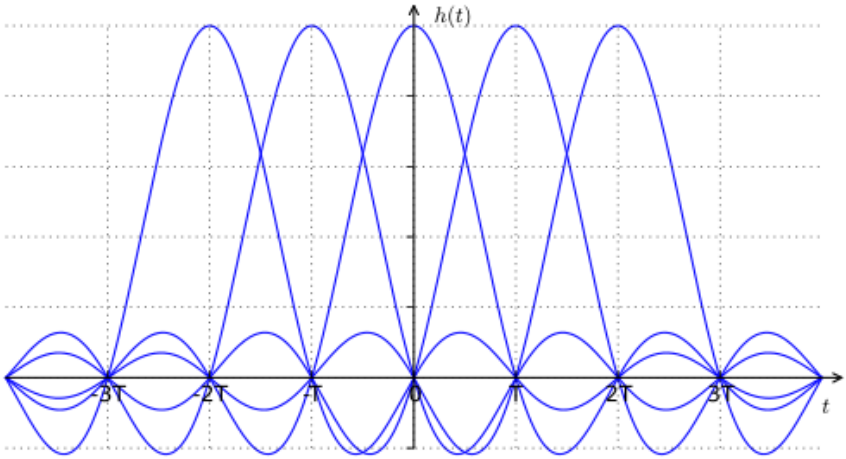


Figure 3.13 Consecutive raised cosine impulses

The value of the roll-off factor is important as it defines the required bandwidth (that is why is it called excess bandwidth). If the raised-cosine impulse is represented in frequency, it can be observed that when the roll-off is zero we have a rectangular pulse, which in time domain is a sinc(x) function, this means more time is required for the function to become zero.

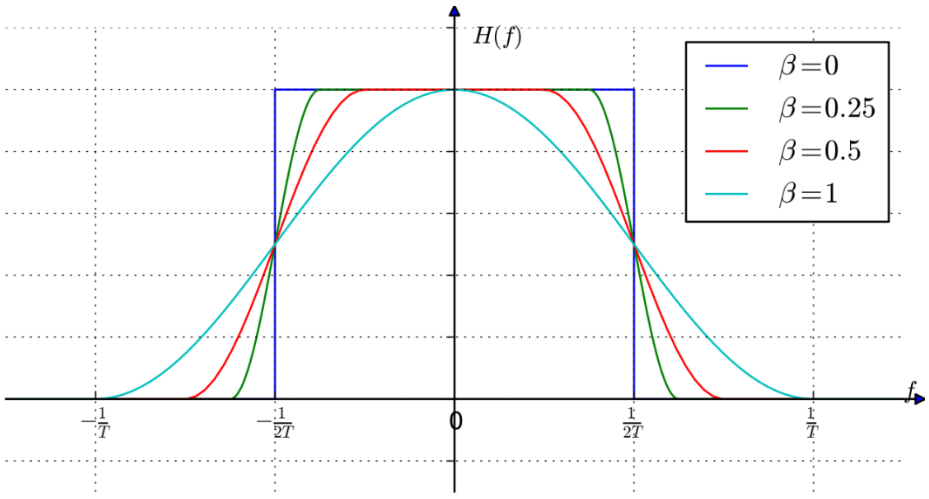


Figure 3.14 Difference between some roll-off values

The implementation of a RRC filter in MATLAB can be done with the aid of the function `rcosdesign`. The required parameters are:

- The roll-off factor (0.35 for TETRA signals).
- The span, which is how much the filter is truncated in terms of symbols duration(8 is good enough).
- The samples per symbols at the input of the filter (4 in this case).
- The last option let us to choose between a normal raised cosine or root raised cosine (`'normal'` or `'sqrt'`).

Then the command in MATLAB® to create the filter coefficients will be:

```
mf = rcosdesign(beta,span,mf_sps,'sqrt');
```

To check the correct behavior of the filter in terms of the required bandwidth of the signal, a representation of the magnitude response can be plot with the sampling frequency set to 72 kHz, which is the sampling frequency at this stage.

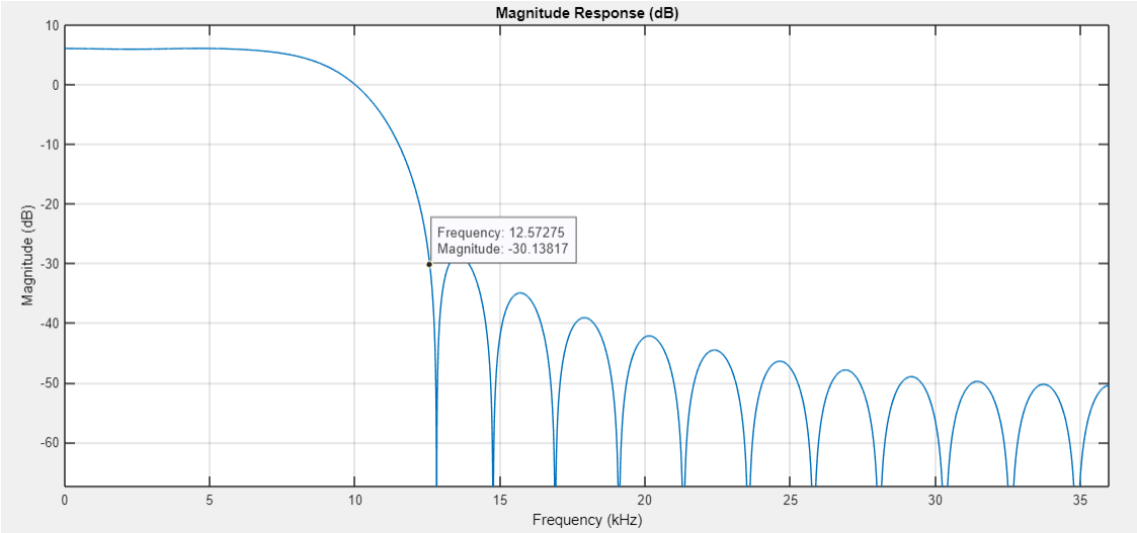


Figure 3.15 Magnitude response of the matched filter.

The Figure 3.14 shows the maximum bandwidth allowed by the matched filter, which is approximately 12.5 kHz (25 kHz due to the symmetry), that correspond to the TETRA signal bandwidth which is 25 kHz.

By plotting the impulse response of the matched filter, we can appreciate the RRC shape as well the length of the filter in coefficients, which has a span of 8 symbols and each symbol has 4 samples, then 8*4=32 coefficients.

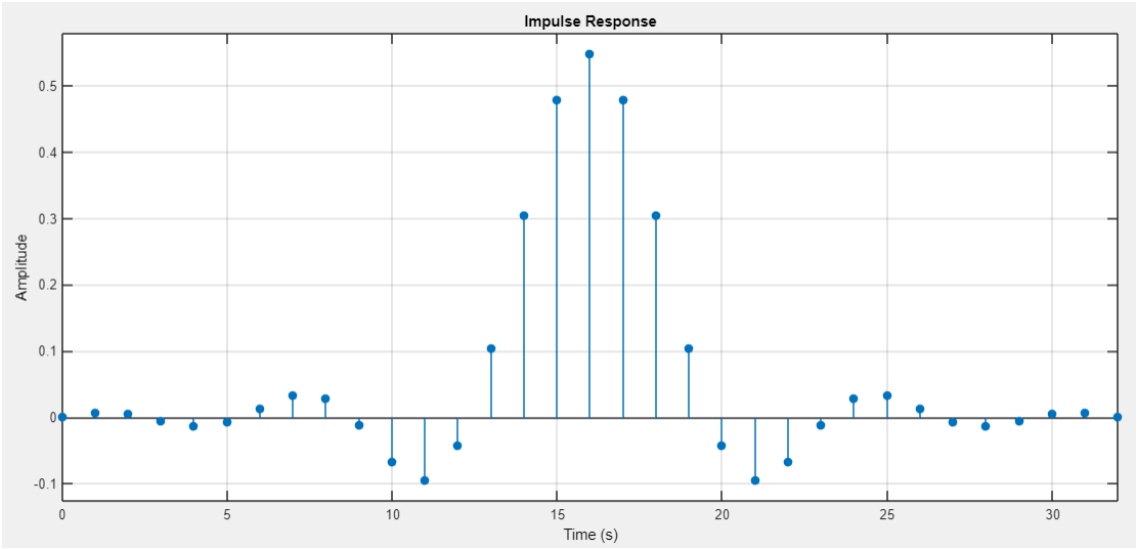


Figure 3.16 Impulse response of the matched filter

As we have 4 samples per symbol at the input of the matched filter, a downsample is necessary to recover the symbols correctly, but since timing recovery stage needs the signal to be delivered at 2 samples per symbol, the signal will be downsampled only by 2.

3.4 MATLAB® RESULTS

In the first stage the received framed is downsampled by a factor of 25 with a CIC filter. The input is the previously received frame `rcv`, which is sampled by the SDR at 2.4 MHz, and the output is a signal with a sampling frequency of 96 kHz.

```
out_CIC_down=filter(CIC_down_25_o2,1,rcv);       %CIC filtering
downsample_25=downsample(out_CIC_down,25);       %downsample by 25
```

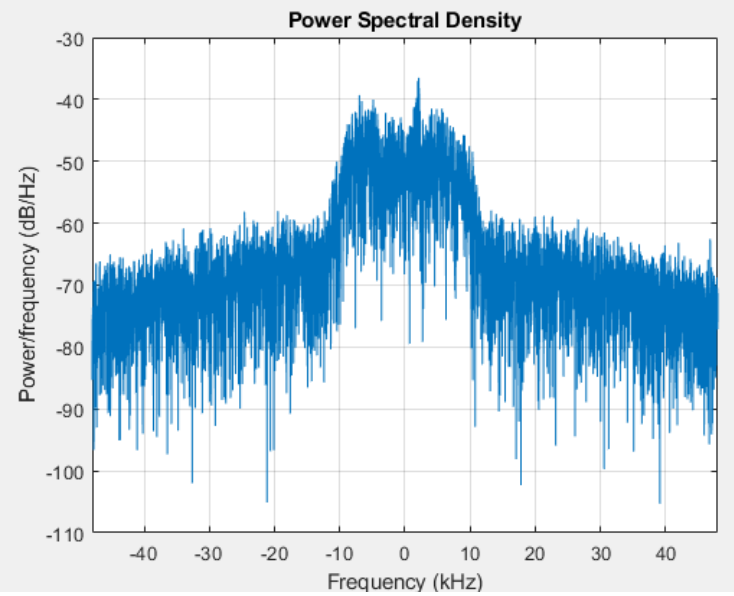


Figure 3.17 Received frame downsampled by 25

Next stage is the upsampling CIC, which is required because we are resampling by a fractional value (100/3). The new sampling frequency is 288 kHz.

```
upsample_3=upsample(downsample_25,3);            %upsample by 3
out_CIC_up=filter(CIC_up_3_o2,1,upsample_3);     %CIC filtering
```

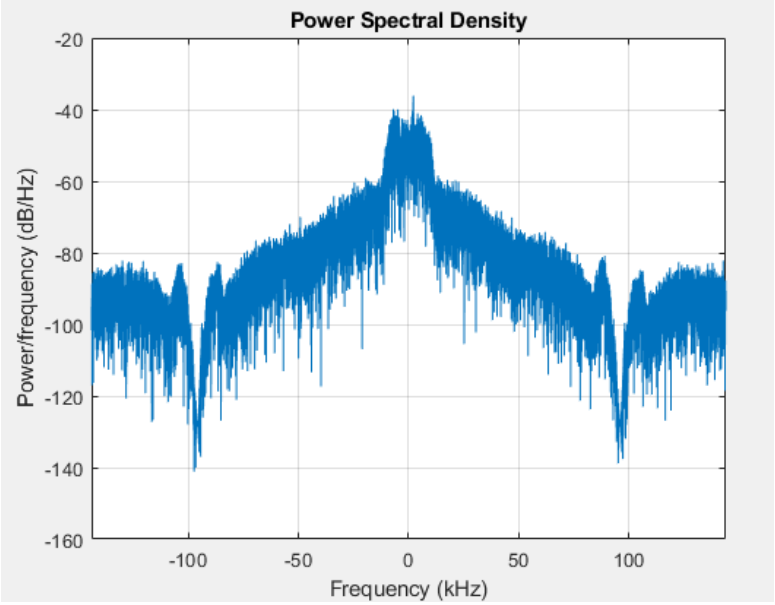


Figure 3.18 Upsampling with CIC filter by a factor of 3

The third stage is the ¼-th band filter downsampling, which compensate the CIC non-flat passband. The sampling frequency of the resulting signal is 72 kHz.

```
out_h4bd = filter(H4bd,1,out_CIC_up);      %1/4 band filtering
downsample_4=downsample(out_h4bd,4);         %downsample by 4
```

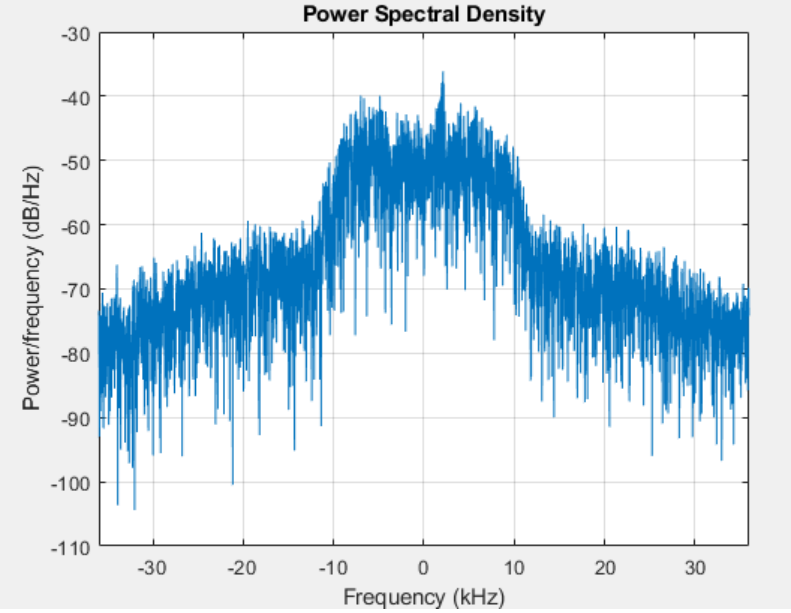


Figure 3.19 ¼-th band downsampling

The last stage is the matched filter, which purpose is to optimize the SNR.

```
out_MF = filter(mf,1,downsample_4);          %matched filter
downsample_2=downsample(out_MF,2);           %downsample to get 2 sps
```

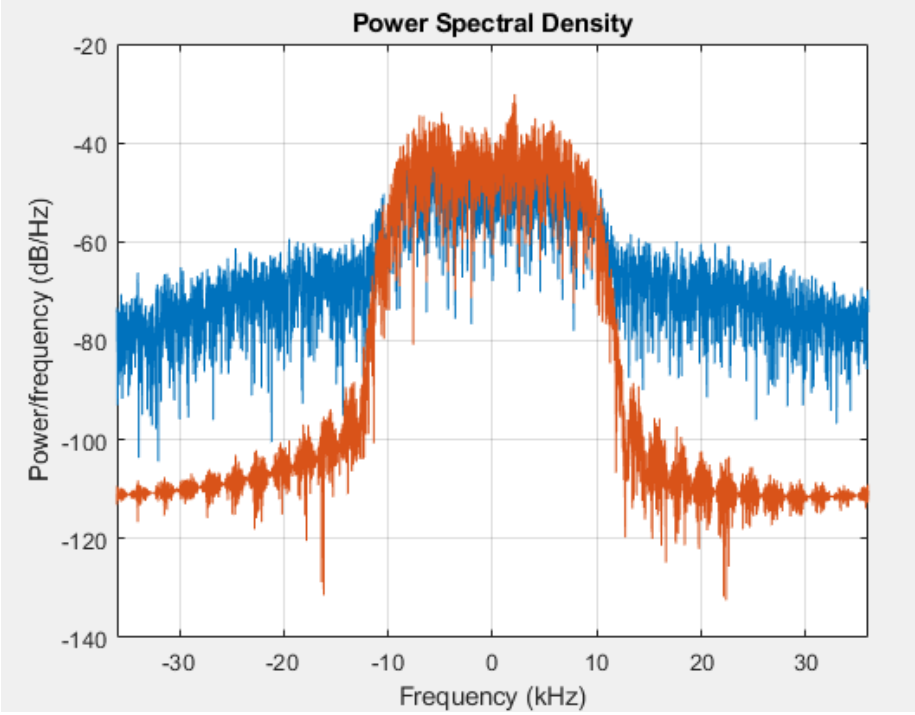


Figure 3.20 Signal before and after matched filtering

In the figure 3.20 we can see the effects of matched filtering. The blue signal is before and the red is after filtering. The SNR increments after filtering.

This signal should be then downsampled by 4 to get 18 kHz, which is the symbol rate for TETRA signals, but since we need 2 samples per symbol for the timing synchronization stage, the downsample factor will be 2.

In the Figure 3.21 we can see how the output of the matched filter would look after downsampling by 4, so these are the received symbols before synchronization stage.

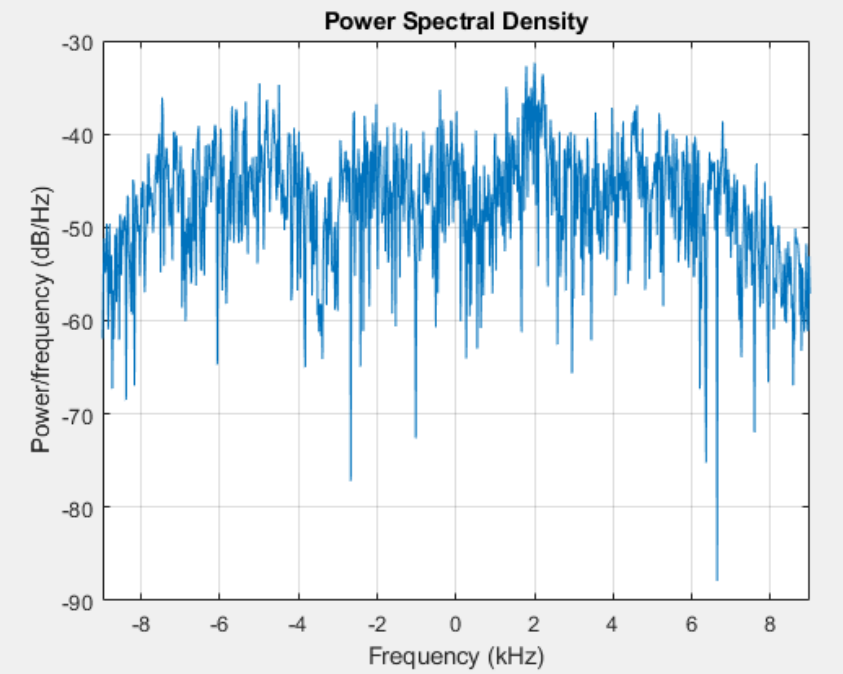


Figure 3.21 Received TETRA symbols

4. TIMING SYNCHRONIZATION

As the signal is sampled at an arbitrary time, the obtained values will differ from the values at the optimal sampling instant, leading to errors in the detection of the symbols. This will also cause ISI, as the pulse shape is not zero out of the optimal sampling instant. The concept is easy to understand with a graphic example:

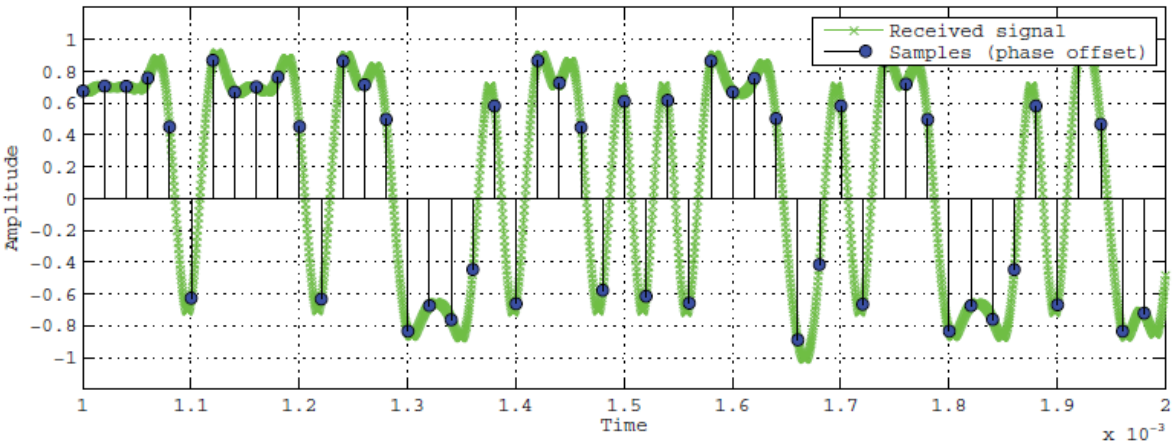


Figure 4.1 Samples with timing offset

In the Figure 4.1 the samples have an offset, falling in time instants where the value is not optimal, and this can lead to a symbol error detection. The modulation used in the figure is BPSK, so it can be easily determined the expected symbol, as only two possible options exist (+1 and -1). With higher order modulations it get more complicated, as the value obtained at the sampling instant will fall into the boundaries of an adjacent symbol, leading to detect the wrong symbol.

The task of the timing synchronization is to estimate the offset in the sampling time and resample the signal to get the correct sampling instant. In the Figure 4.2 the same signal is shown but with the samples taken in the maximum effect points. This is what we expect to achieve with the timing synchronization stage.

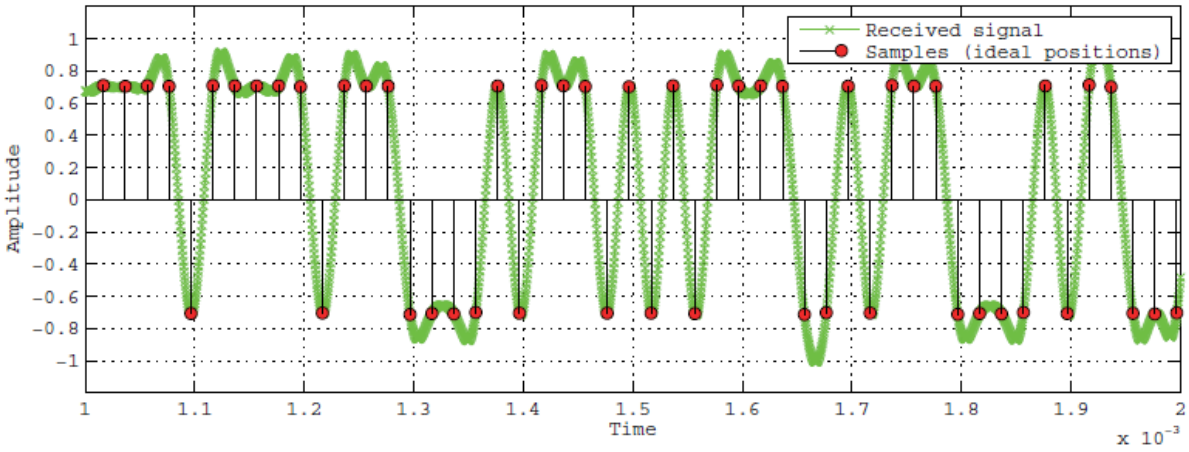


Figure 4.2 Samples taken at ideal positions

4.1 TIMING RECOVERY BLOCK STRUCTURE

The timing recovery block is based on a Phase Locked Loop (PLL) structure. A general block diagram with the main components is presented in the Figure 4.3.

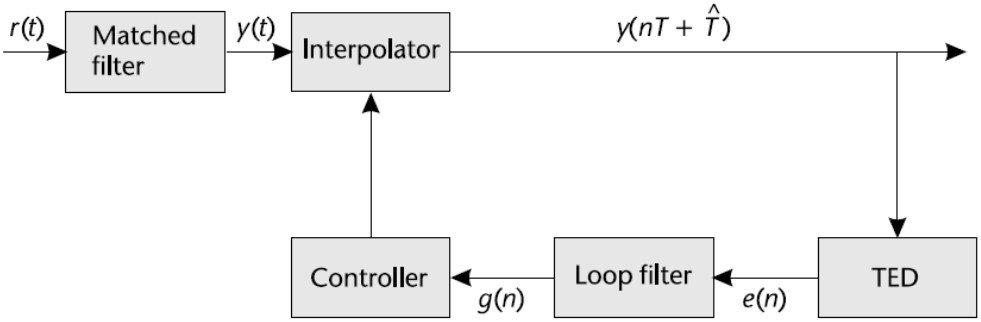


Figure 4.3 General structure of a PLL based timing recovery

The timing recovery stage is located after the matched filter. The samples delivered by the matched filter pass through an interpolator, which is driven by the controller through the output of the loop filter, which is at the same time dependant of the error signal calculated by the Timing Error Detector (TED). This means the interpolation factor is time variable, and the main goal of this block, is to keep the error signal e(n) near to zero, which is the locked position.

4.1.1 INTERPOLATOR FILTER

To understand the interpolation stage let us suppose we have a signal with samples separated by the time T:

$$x\big((n-1)T\big), x(nT), x\big((n+1)T\big), x((n+2)T)$$

and the optimal sample x(t) is located at:

$$t = kT_I$$

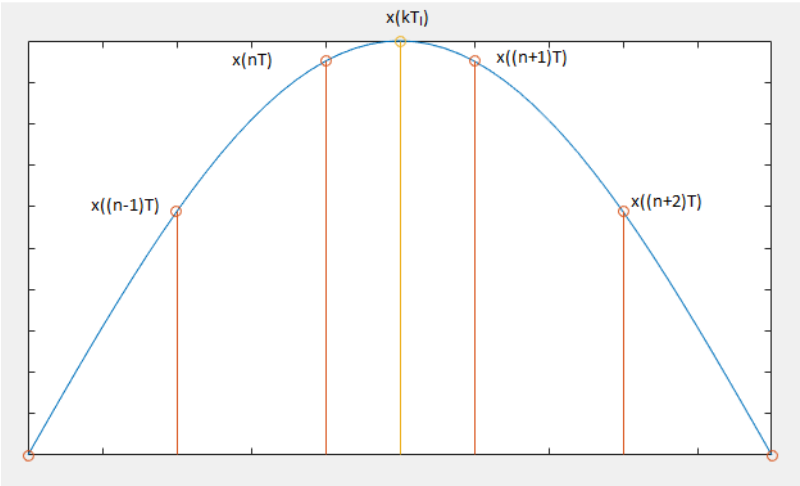


Figure 4.4 Illustration of the obtained samples and the optimal sample

To obtain this optimal sample interpolation is used. When the optimal sample is located between $x(nT)$ and $x\big((n+1)T\big)$, the index n is called $m(k)$, and the fractional interval between $m(k)$ and the time instant kT_I is called $\mu(k)$, which satisfies:

$$0 \leq \mu(k) < 1$$

and is defined by:

$$\mu(k)T = kT_I - m(k)T$$

This concept is illustrated in the Figure 4.5

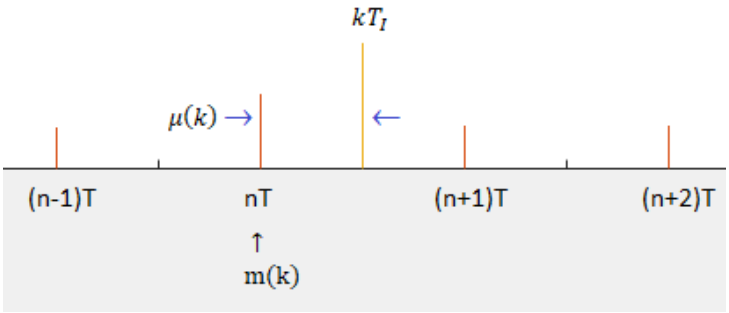


Figure 4.5 Relationship between obtained and desired samples

The value of $\mu(k)$ determines the output of the interpolator filter as it is dynamically updated with every iteration.

The type of filter used for this section is a piecewise polynomial filter. The order of the filter is equivalent to the order of the polynomial used to estimate the received signal, in this case is a quadratic or second order, as it approximates to the desired shape as shown in the Figure 4.6. More accurate implementations exist, as the cubic interpolator, but the implementation gets more complex.

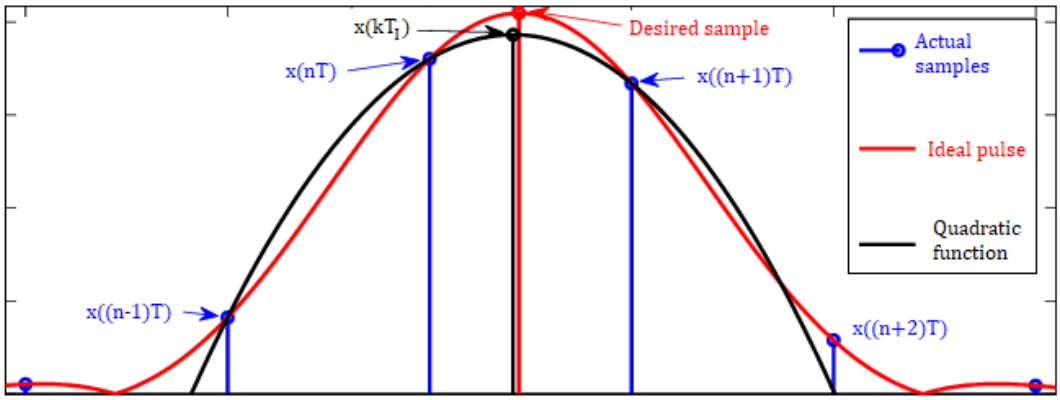


Figure 4.6 Quadratic interpolation

The coefficients of the filter are constant, and they are presented in the table below, where $\alpha = 0.5$ in this case.

i	b2(i)	b1(i)	b0(i)
-2	α	-α	0
-1	-α	1+α	0
0	-α	α-1	1
1	α	-α	0

The polynomial that implements the function $x(t)$ can be defined as:

$$x(t) \approx c_p t^p + c_{p-1} t^{p-1} + \cdots + c_1 t + c_0$$

Then the interpolant at $t = kT_I = (m(k) + \mu(k))T$ is obtained from:

$$x(t) \approx c_p(kT_I)^p + c_{p-1}(kT_I)^{p-1} + \cdots + c_1(kT_I) + c_0$$

As we are using a second order polynomial $p = 2$, the polynomial approximation will be:

$$x(t) \approx c_2 t^2 + c_1 t + c_0$$

To compute the desired interpolant at $t = (m(k) + \mu(k))T$, we make the substitution:

$$x((m(k) + \mu(k))T) \approx c_2 \Big((m(k) + \mu(k))T\Big)^2 + c_1((m(k) + \mu(k))T) + c_0$$

This requires three samples, implying filter will not be symmetric with respect to $\mu(k) = \frac{1}{2}$ and using it will produce a discontinuity that is easier to understand with the Figure 4.7, where we can see how a discontinuity appears between every pair of samples.

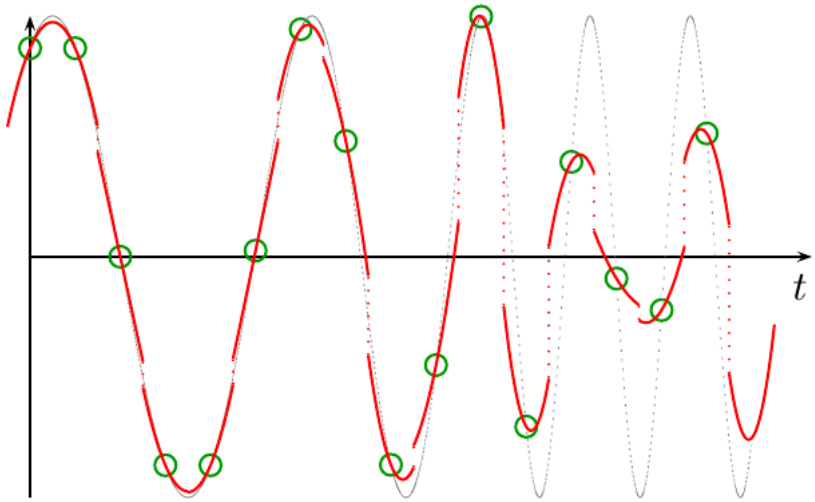


Figure 4.7 Discontinuity with 3 samples interpolator

The reason behind this, is that the frequency response of this quadratic interpolator isn't continuous and for this reason a filter with four taps will be used instead.

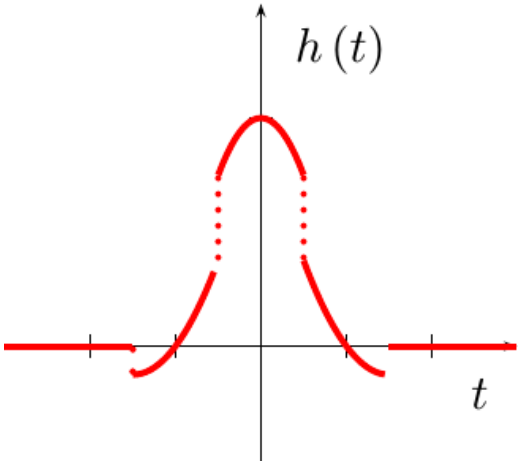


Figure 4.8 Impulse response of the quadratic interpolator

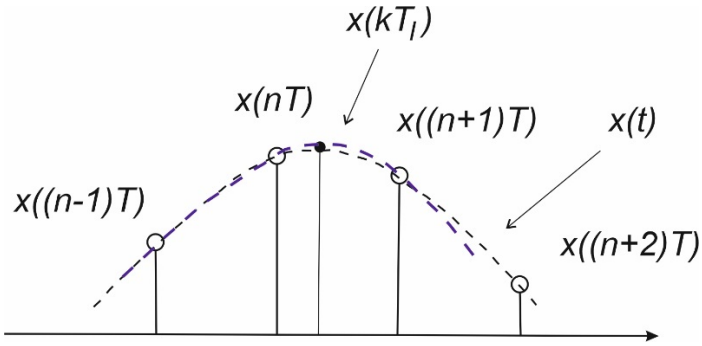


Figure 4.9 Three samples second order filter

The interpolant can be obtained from a filter of the form:

$$x((m(k)+\mu(k))T) = \sum_{i=-2}^{1} h_2(i)x((m(k)+\mu(k))T)$$

And the filter coefficients, based on Table 4.1, would be:

$$h_2(-2) = \alpha\,\mu(k)^2 - \alpha\mu(k)$$

$$h_2(-1) = -\alpha\,\mu(k)^2 - (1+\alpha)\mu(k)$$

$$h_2(0) = -\alpha\,\mu(k)^2 - (1-\alpha)\mu(k) + 1$$

$$h_2(1) = \alpha\,\mu(k)^2 - \alpha\mu(k)$$

The value $\alpha = 0.5$ is selected as it has been proven to reduce the complexity of the hardware.

Using a polynomial interpolator results in:

$$x((m(k)+\mu(k))T) = \sum_{i=-I_1}^{I_2} h_p(i;\mu(k))x\big((m(k)-i)T\big) \qquad (1)$$

Each filter coefficient $h_p(i;\mu(k))$ could be expressed as a polynomial in $\mu(k)$:

$$h_p\big(i;\mu(k)\big) = \sum_{l=0}^{p} b_l(i)\mu(k)^l \qquad (2)$$

By substituting (2) into (1) and rearranging:

$$x((m(k)+\mu(k))T) = \sum_{l=0}^{p} \mu(k)^l \sum_{i=I_1}^{I_2} b_l(i)x((m(k)-i)T) \qquad (3)$$

If we define:

$$v(l) = \sum_{i=I_1}^{I_2} b_l(i)x((m(k)-i)T)$$

It can be observed that the expression $v(l)$ represents a filter equation, where samples $x((m(k)-i)T)$ pass through the filter $b_l(i)$, and its coefficients are fixed and independent of $\mu(k)$.

By using nested evaluation in the expression (3) and expressing it in terms of $v(l)$, we have:

$$x((m(k)+\mu(k))T) = \big(v(2)\mu(k)+v(1)\big)\mu(k)+v(0)$$

The code that implements the filter in MATLAB® is the following:

```
v2 = 1/2*[1, -1, -1, 1]*x(n:-1:n-3);    % Farrow structure for the
v1 = 1/2*[-1, 3, -1, -1]*x(n:-1:n-3);   % piecewise parabolic
v0 = x(n-2);                            % interpolator
xI = (mu*v2 + v1)*mu + v0;              % Interpolator output
```

The implemented filter is delayed 2 samples in order to operate on $[x(nT), x\big((n-1)T\big), x\big((n-2)T\big), x\big((n-3)T\big)]$, so it will be a causal operation, and it creates an interpolant between $x\big((n-1)T\big), x\big((n-2)T\big)$.

4.1.2 TIMING ERROR DETECTOR (TED)

The TED generates an error signal based on the input and output of the matched filter, meaning it is generated at the symbol rate.

There are many algorithms for the TED block, but in this project, Gardner TED will be used. It works at 2 samples per symbol, is non-data aided and doesn't require frequency correction, so it can be used before the carrier frequency recovery stage.

The Gardner TED is based on finding the zero crossing in the eye diagram. Assuming the matched filter output at 2 samples per symbol and index (k), we have:

$$\dots x\big((k-1)T_s-\tau\big), x\left(\left(k-\frac{1}{2}\right)T_s-\tau\right), x(kT_s-\tau), x\left(\left(k+\frac{1}{2}\right)T_s-\tau\right)\dots$$

The error signal for the Gardner TED is then defined by:

$$e(k) = x\left(\left(k-\frac{1}{2}\right)T_s-\tau\right)\big[x\big((k-1)T_s-\tau\big) - x(kT_s-\tau)\big]$$

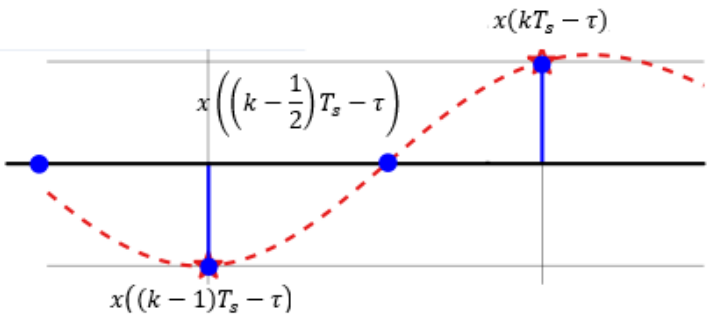


Figure 4.10 Illustration for the Gardner TED interpretation

We can think of the error signal as:

$$e = Actual\ sample * (Late\ sample - Early\ sample)$$

And the goal is to force the middle sample to approach the zero crossing.

The implementation of the Gardner TED in MATLAB® is:

```
eR=real(TEDBuff(1)) * (real(TEDBuff(2)) - real(xI)); %Gardner real
```

```
eI=imag(TEDBuff(1)) * (imag(TEDBuff(2)) - imag(xI)); %Gardner imaginary
e=eR+eI;                                             %Gardner sum
```

As we are working with a signal which has imaginary and real parts, the error signal is calculated for each branch and then the resulting error is the sum of both.

`TEDBuff` is an array containing the previous samples of the signal, and `xI` is the actual interpolator output.

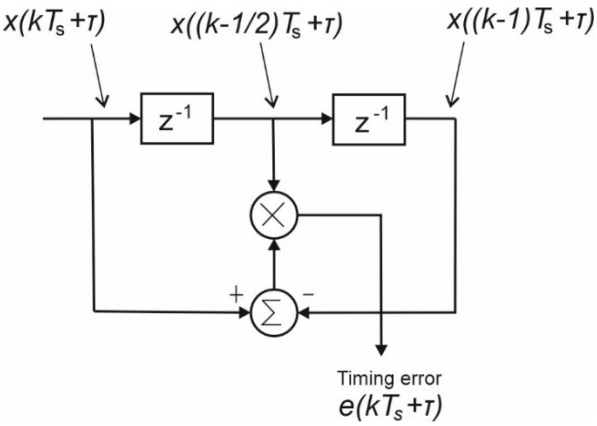


Figure 4.11 Gardner block diagram

4.1.3 LOOP FILTER

One of the main functions of the PLL loop filter is to determine the stability. This has to do with the response of the loop to the changes in the reference input, the range where the lock state is achieved (pull-in range) and how fast it does (lock time). By doing this, a suitable control signal can be delivered to the controller.

The secondary function is to suppress high frequency components and noise. A Proportional-Integrator (PI) loop filter is used for this purpose.

The proportional part is the error signal of the TED with a gain K1, while the integrator is an ideal integrator of the error signal with gain K2.

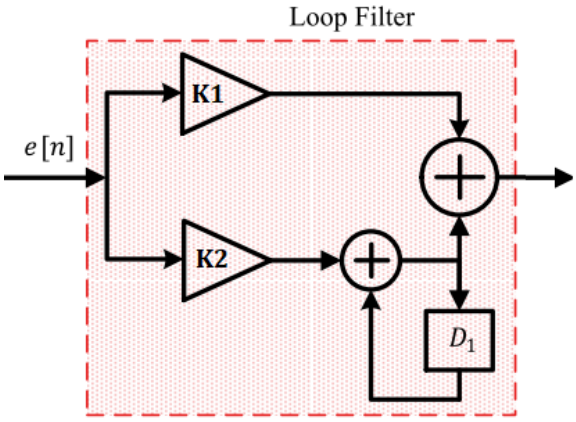


Figure 4.12 Proportional-Integrator loop filter

The implementation of the loop filter in MATLAB® is simple:

```
vp = K1*e;          % proportional component of loop filter
vi = vi + K2*e;     % integrator component of loop filter
```

```
v = vp + vi;        % loop filter output
```

The values of K_1 and K_2 can be obtained from the classical equations for PLL design, but before the gain K_P must be calculated. The S-curve for the Gardner algorithm is defined by:

$$g(\tau_e) = \frac{4K^2E_{avg}}{T_s} * \frac{1}{4\pi\left(1-\frac{\alpha^2}{4}\right)} * \sin\left(\frac{\pi\alpha}{2}\right) * \sin\left(2\pi\frac{\tau_e}{T_s}\right)$$

With $\frac{K^2E_{avg}}{T_s} = 1$ and α=0.35, the S-curve will look like:

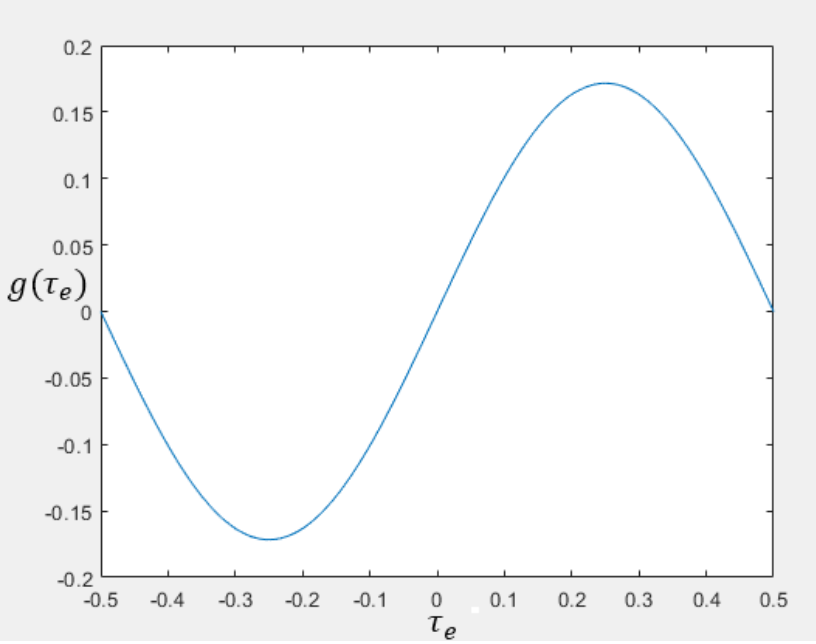


Figure 4.13 S-curve for the Gardner TED

Then K_P is defined as the gain of the S-Curve:

$$K_P = \frac{4K^2E_{avg}}{T_s} * \frac{1}{4\pi\left(1-\frac{\alpha^2}{4}\right)} * \sin\left(\frac{\pi\alpha}{2}\right)$$

Now, we must choose a damping factor. The ranges of the damping factor are:

$$\zeta = \begin{cases} < 1, Underdamp \\ = 1, Critically\ damped \\ > 1, Overdamped \end{cases}$$

When the damping factor is large, the convergence is faster, while smaller values produce a long period of transient behaviour with overshoots before converging, but as an advantage, smaller values have a better ability to track changes. We will use the value $\zeta = \frac{1}{\sqrt{2}}$.

Another required value is $B_N T_S$ which is the noise bandwidth multiplied by the symbol period and is defined as 0.005, and the value K_0= -1. The selection of the negative K_0 is related to the controller section, where a decrescent counter is implemented.

Once we have these values, we can proceed with the estimation of K_1 and K_2. These two can be obtained from the equations:

$$K_0 K_P K_1 = \frac{\frac{4\zeta}{N}\left(\frac{B_N T_S}{\zeta + \frac{1}{4\zeta}}\right)}{1 + \frac{2\zeta}{N}\left(\frac{B_N T_S}{\zeta + \frac{1}{4\zeta}}\right) + \left(\frac{B_N T_S}{N\left(\zeta + \frac{1}{4\zeta}\right)}\right)^2}$$

$$K_0 K_P K_2 = \frac{\frac{4\zeta}{N^2}\left(\frac{B_N T_S}{\zeta + \frac{1}{4\zeta}}\right)^2}{1 + \frac{2\zeta}{N}\left(\frac{B_N T_S}{\zeta + \frac{1}{4\zeta}}\right) + \left(\frac{B_N T_S}{N\left(\zeta + \frac{1}{4\zeta}\right)}\right)^2}$$

4.1.4 INTERPOLATION CONTROL

This block is responsible of controlling the interpolation filter by providing the $m(k)$index and the fractional interval $\mu(k)$. The basepoint index $m(k)$ is identified by a signal often called a strobe.

The method used here is a modulo-1 Counter, which means it will underflow every N samples, that is when a new symbol start, and the underflow is aligned with the basepoint indexes $(m(k))$.

The output of the loop filter adjusts the amount by which the counter decrements in order to align the underflows with the sample time of the desired interpolant. The underflow is signalled by the strobe, and it's used by the interpolator to identify the sample time of the interpolant $m(k)$.

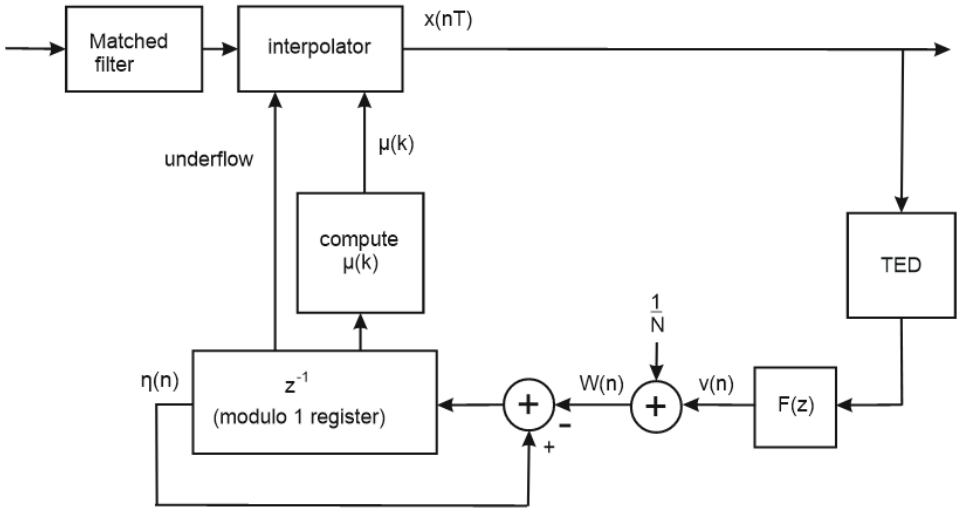


Figure 4.14 Complete structure of timing synchronization block

As we want to produce a trigger every N samples, this will occur when the output of the loop filter $v(n)$ is zero (locked state), so the decrement would be:

$$W(n) = v(n) + \frac{1}{N}$$

Leading to a maximal value of 1 under modulo-1 subtraction of the counter:

$$\eta(n+1) = \big(\eta(n) - W(n)\big) \quad Mod.-1$$

Before the counter is updated, a trigger condition is checked:

$$Trigger = \begin{cases} \eta(n) < W(n) & true \\ otherwise & false \end{cases}$$

This triggering signal indicates the start of a new symbol and is used also to assure we are estimating error over the correct samples.

When the trigger occurs, the fractional interval $\mu(k)$ is updated, and is dependant of the counter step and the current count:

$$\mu(k) = \frac{\eta(n)}{W(k)}$$

This fractional interval is then passed to the interpolator, so it will apply the required delay.

The Figure 4.15 is useful to understand all the concepts introduced before and is helpful to understand the relationship between the available samples, the desired interpolants and the modulo-1 counter. The trigger occurs every N samples, that is, at the indexes of $m(k)$, this occur before the desired interpolants at the indexes of kT_I, and then the factional delay $\mu(k)$ is sent to the interpolator, so the signal is delayed according to obtain the desired interpolant.

And for interpretation of the counter, if we define it in terms of $m(k)$ and incorporate the modulo-1 reduction, we have:

$$\eta(m(k)+1) = 1 + \eta(m(k)) - W(m(k))$$

Rearranging to put the terms of the current count and the step count on one side:

$$1 - \eta(m(k)+1) = \eta(m(k)) - W(m(k))$$

Another aspect we must take in count in the interpolation block is that the sampling frequency is not always the same, sometimes it can be faster or slower than the 2 required samples causing a timing error, this will produce two possible situations:

1) $T > \frac{T_S}{2}$: in this case there will be some instants where the interpolant will be generated one sample apart instead of two samples. This occurs when the accumulated timing error exceeds one sample period at the same time when $\mu(k)$ is decreasing to zero and wrapping around one, therefore one interpolant will not be produced, so we must insert this interpolant. Here the underflow will be 1 for two consecutive samples.
2) $T < \frac{T_S}{2}$: this is the opposite case, there are instants where the interpolant will be generated three samples apart instead of two samples, there will be an extra interpolant produced which should be discarded. As the first case, this happens when the accumulated timing error exceeds one sample period but at the same time when $\mu(k)$ is exceeding one and wrapping around zero. Underflow value is 0 for two consecutive samples.

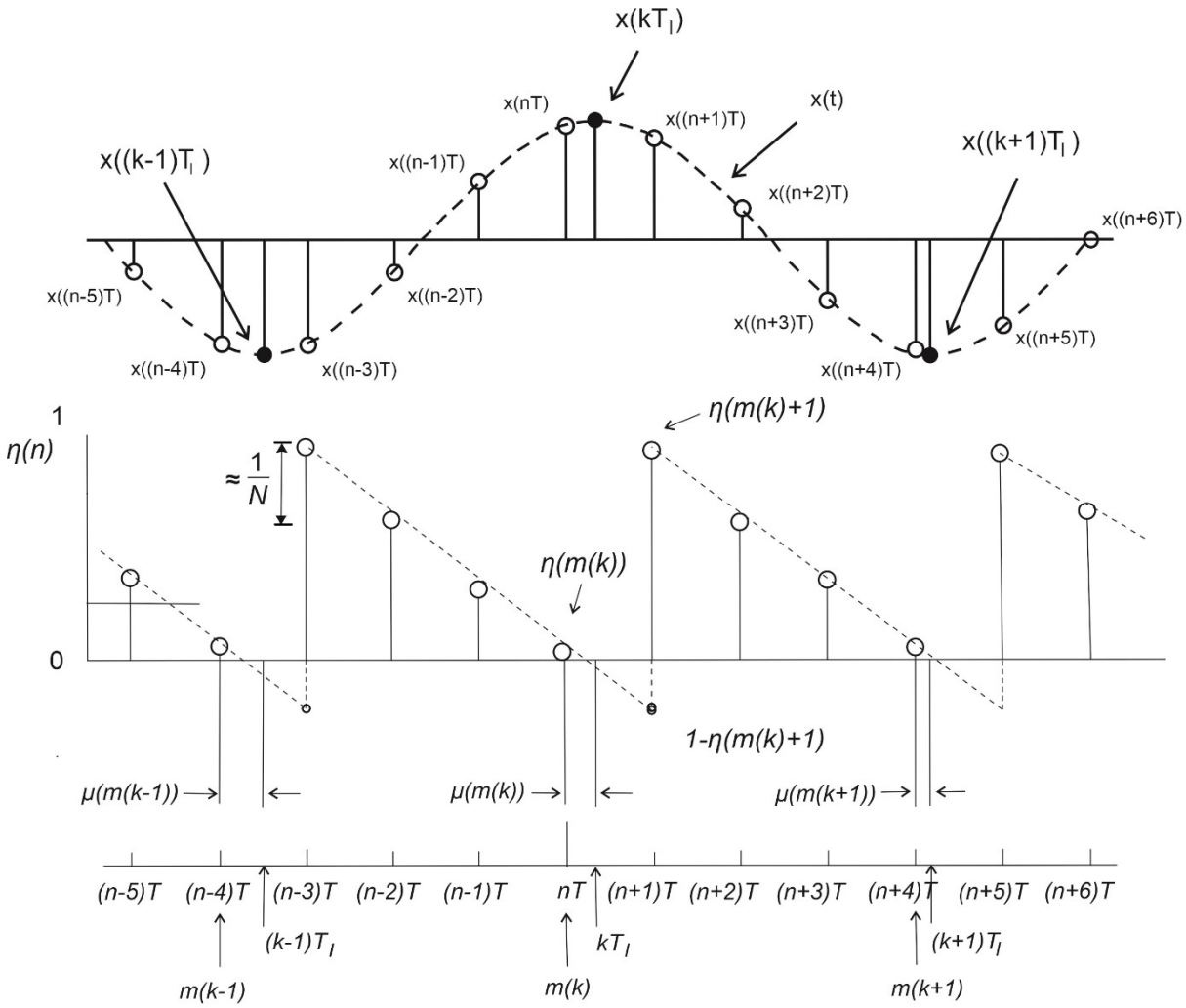


Figure 4.15 Available samples, desired interpolants and counter contents

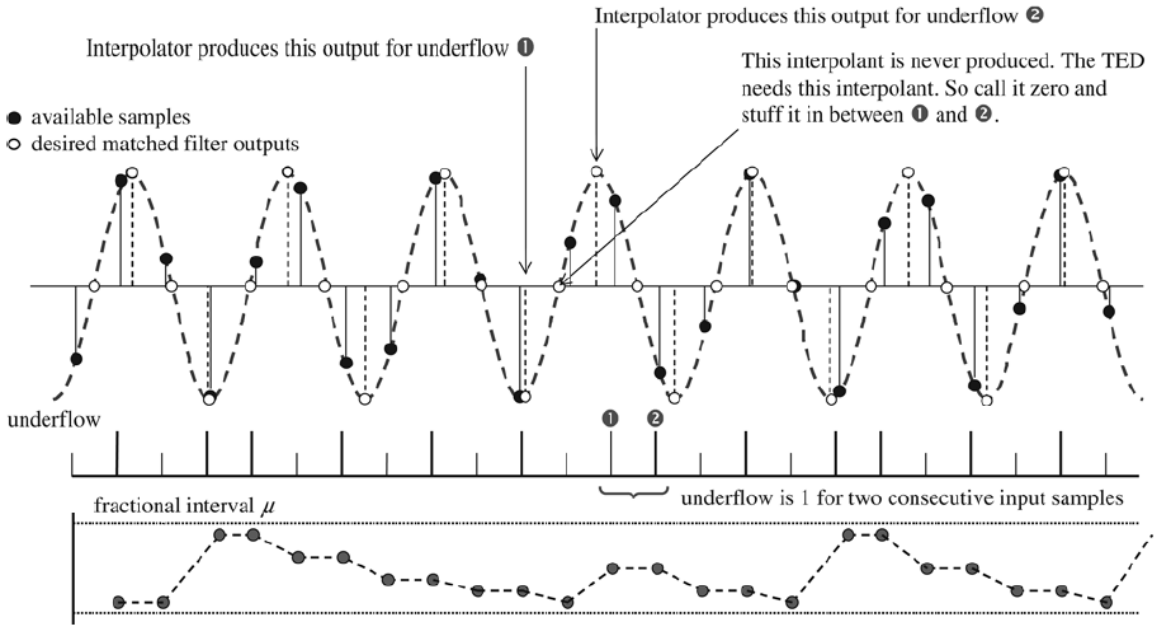


Figure 4.16 Missing interpolant when $T > \frac{T_S}{2}$

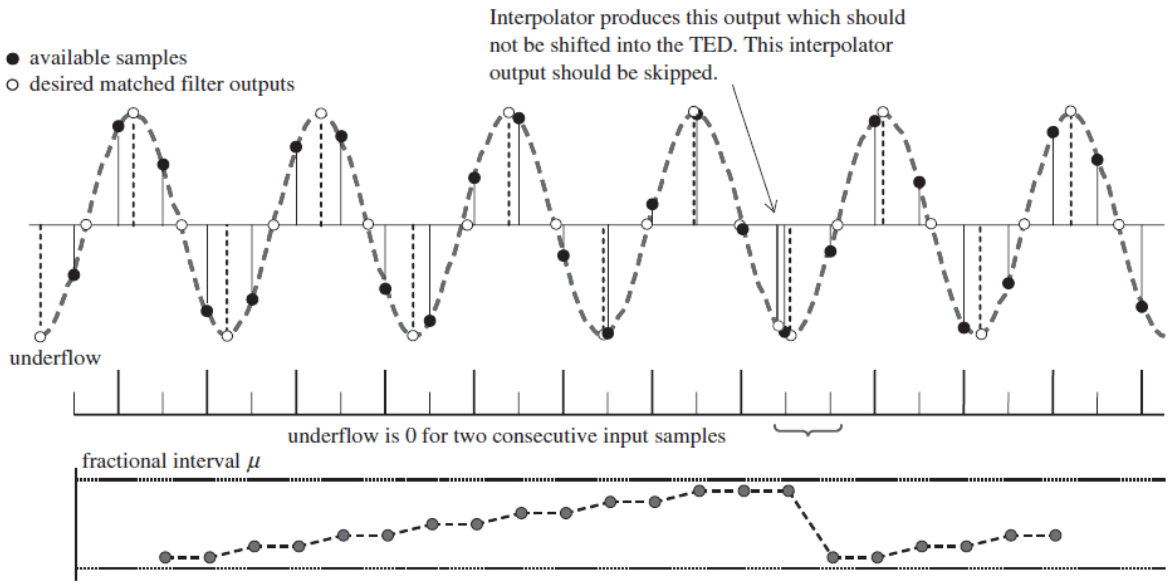


Figure 4.17 Extra interpolant when $T < \frac{T_S}{2}$

The code to implement this interpolation controller in MATLAB® is:

```
W = 1/N + v; % NCO control word
% update registers
    if underflow == 0 && old_underflow == 0
        TEDBuff = TEDBuff;              % skip current sample
    elseif underflow == 0 && old_underflow == 1
        TEDBuff = [xI; TEDBuff(1)];     % normal operation
    elseif underflow == 1 && old_underflow == 0
        TEDBuff = [xI; TEDBuff(1)];     % normal operation
    elseif underflow == 1 && old_underflow == 1
        TEDBuff = [xI; 0];              % stuff missing sample
    end
CNT_next = CNT - W;              % update counter value for next cycle
     if CNT_next < 0             % test to see if underflow has occurred (CNT<W)
     CNT_next = 1 + CNT_next;    % reduce counter modulo-1 if underflow
     old_underflow = underflow;
     underflow = 1;              % set underflow flag
     mu_next = CNT/W;            % update mu
        else
        old_underflow = underflow;
        underflow = 0;
        mu_next = mu;
     end
```

4.2 MATLAB® SIMULATION

After downsampling the output of the matched filter by 4, we will obtain the received symbols, but they are not aligned with the ideal sampling instant, so we see a dispersed constellation, and because the carrier is not yet compensated, the symbols constantly rotate creating a circle.

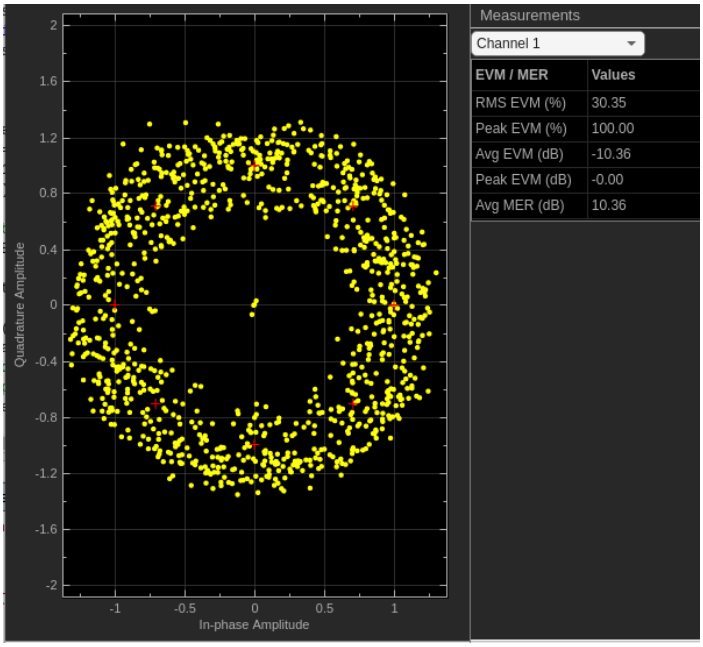


Figure 4.18 Received symbols without timing and frequency recovery

After the timing recovery stage, the symbols are less dispersed, but still misplaced due to the lack of frequency synchronization.

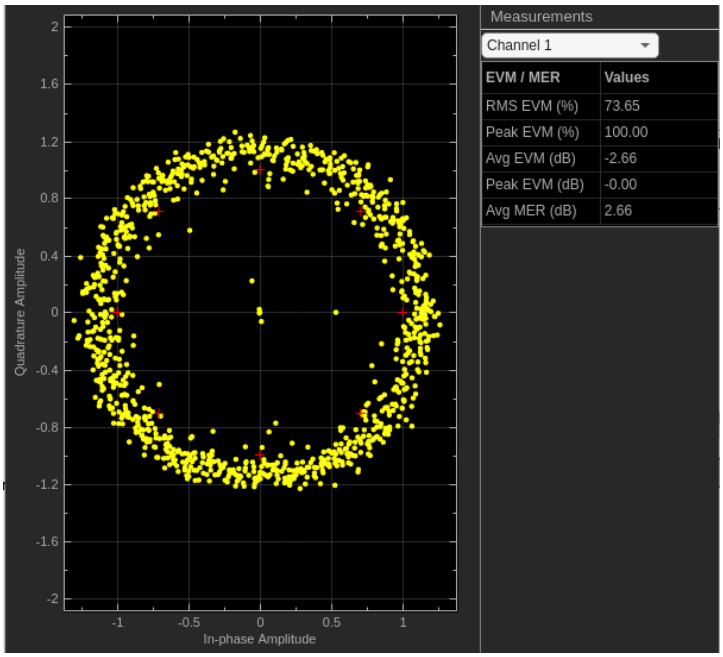


Figure 4.19 Symbols after timing recovery

If we plot the values of μ and the estimated error by the Gardner algorithm, we can see the relationship between them both. When the estimated error is high, also the μ is high, meaning a longer distance from optimal sampling instant, then it reaches a lock point where both values tend to zero.

Figure 4.20 Relationship between the estimated error and μ

5. FREQUENCY SYNCHRONIZATION

The local oscillator (LO) used in the transmitter and the one used in the receiver will have offsets due to multiple factors as the temperature, electrical noise among others, this leads to a mismatch between the transmitter and receiver carrier frequencies, causing a rotation in the constellation and making difficult to correctly identify the received symbols.

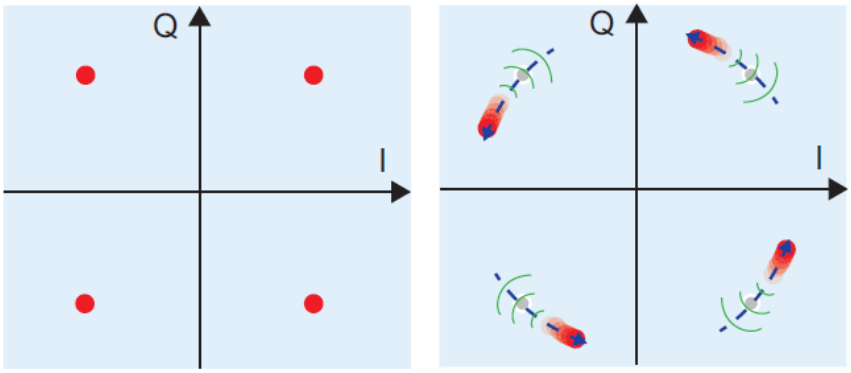


Figure 5.1 Ideal QPSK constellation and effects of frequency offset

To perform the carrier synchronization, a loop can be used to create a sinusoidal wave with same frequency and phase as the carrier used in the received signal. This is achieved with the aid of a PLL which is used to track changes in time-varying phase. When in lock status, the frequency offset estimated by the PLL can be used to compensate the carrier offset.

Many algorithms for carrier recovery exist, but one had an enormous impact in the signal processing field: the Costas loop.

5.1 COSTAS LOOP

In the past it was mandatory to send data along with a pilot tone for carrier synchronization, which was high power demanding. John Costas demonstrated that carrier could be recovered from received signal without a pilot tone. The Costas loop for QPSK will be used in this project.

To understand how it works, we will refer to the block diagram presented in the Figure 5.2.

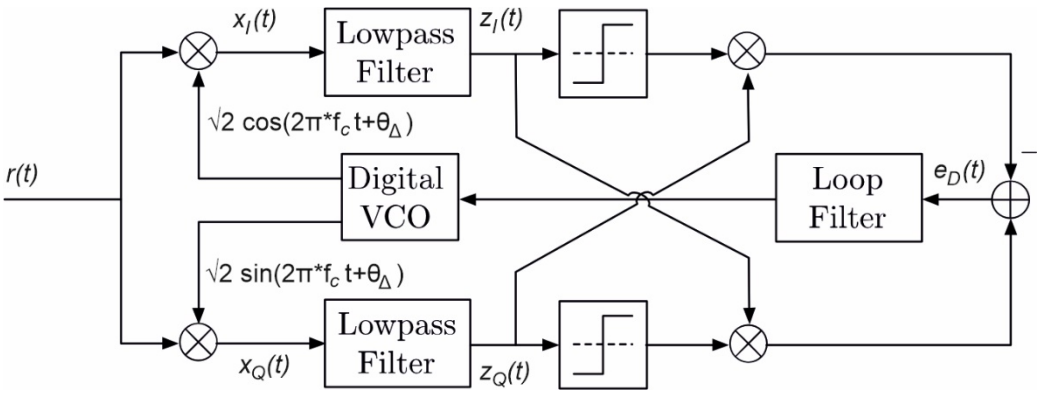


Figure 5.2 Costas loop for QPSK

Let's assume we have in the input a QPSK signal $r(t)$ with the carrier and a phase offset θ_e:

$$r(t) = v_I(t)\sqrt{2}\cos(2\pi f_c t + \theta_e) + v_Q(t)\sqrt{2}\cos(2\pi f_c t + \theta_e)$$

The signal is then multiplied in both branches (I and Q) by the sinusoids produced by the VCO. In the case of the I branch we have:

$$x_I(t) = r(t)\sqrt{2}\cos\left(2\pi f_c t + \theta_\Delta\right)$$

$$x_I(t) = \left\{v_I(t)\sqrt{2}\cos(2\pi f_c t + \theta_e) + v_Q(t)\sqrt{2}\cos(2\pi f_c t + \theta_e)\right\} * \sqrt{2}\cos\left(2\pi f_c t + \theta_\Delta\right)$$

We can use the following trigonometrical identities:

$$\cos(A) * \cos(B) = 0.5\{\cos(A - B) + \cos(A + B)\}$$

$$\sin(A) * \cos(B) = 0.5\{\sin(A + B) + \sin(A - B)\}$$

And then we will have:

$$x_I(t) = v_I(t)\{\cos(2\pi f_c t + \theta_e - 2\pi f_c t - \theta_\Delta) + \cos(2\pi f_c t + \theta_e + 2\pi f_c t + \theta_\Delta)\} +$$

$$v_Q(t)\{\sin(2\pi f_c t + \theta_e + 2\pi f_c t + \theta_\Delta) + \sin(2\pi f_c t + \theta_e - 2\pi f_c t - \theta_\Delta)\}$$

After passing through the low pass filter, the double frequency component is removed:

$$z_I(t) = v_I(t)\{\cos(\theta_e - \theta_\Delta) + \cancel{\cos(4\pi f_c t + \theta_e + \theta_\Delta)}\} +$$

$$v_Q(t)\{\cancel{\sin(4\pi f_c t + \theta_e + \theta_\Delta)} + \sin(\theta_e - \theta_\Delta)\}$$

We can define $\theta_C = \theta_e - \theta_\Delta$, which is the carrier phase error:

$$z_I(t) = v_I(t)\cos(\theta_C) + v_Q(t)\sin(\theta_C)$$

And if we apply same operations for the Q branch, we will arrive to the term:

$$z_Q(t) = -v_I(t)\sin(\theta_C) + v_Q(t)\cos(\theta_C)$$

Assuming the PLL is in tracking mode and thus the phase error $\theta_C = \theta_e - \theta_\Delta$ is small, this will lead to the approximation $\cos(\theta_C) \approx 1$ and $\sin(\theta_C) \approx 0$. Then we have:

$$z_I(t) = v_I(t)$$

$$z_Q(t) = v_Q(t)$$

The next block is called a slicer, and it takes the sign of their input; therefore, the error signal is defined as:

$$e_D(t) = sign\left(z_Q(t)\right) * z_I(t) - sign\left(z_I(t)\right) * z_Q(t)$$

But because of the positive sign of the quadrature signal used in the example, it would become:

$$e_D(t) = sign\left(z_I(t)\right) * z_Q(t) - sign\left(z_Q(t)\right) * z_I(t)$$

It was also assumed that the symbol value is $\sqrt{2}$ for ease of calculation, but it would be in fact the half of the symbol value, meaning the signal is more susceptible to noise.

This error signal will be then delivered to the loop filter, which is responsible of tracking the changes (just as in the timing recovery block) and locking when $\theta_e \approx \theta_\Delta$. After that, the output of the loop filter is passed to the digital VCO, this value is the phase used to synthetize the carrier wave.

5.2 SECOND ORDER PLL

The PLL is responsible of tracking the changes in the reference signal and locking when the error estimate is zero. The order of the PLL defines its behavior. Here a second order PLL will be used, and to obtain the required values is good to review how does it work.

The block diagram of a PLL in discrete time is:

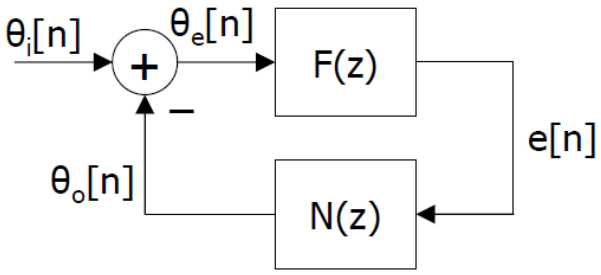


Figure 5.3 Discrete time PLL

Where θ_e is the phase error estimated by the Costas Loop and θ_O is the phase of the local oscillator. When $\theta_e = 0$, the phase of the received signal and the local oscillator will be the same and the PLL will be locked.
N(z) is an integrator which output is:

$$\theta_O[n] = \theta_O[n-1] + e[n-1]$$

The integrator is defined in the Z domain as:

$$N(z) = \frac{z^{-1}}{1 - z^{-1}}$$

If we define $L(z) = F(z)\,N(z)$, the transfer function of this block would be:

$$H(z) = \frac{\Theta_o(z)}{\Theta_i(z)} = \frac{L(z)}{1 + L(z)} = \frac{z^{-1}F(z)}{1 - z^{-1} + z^{-1}F(z)}$$

And the phase error can be obtained from:

$$\Theta_e(z) = \Theta_i(z)\big(1 - H(z)\big) = \Theta_i(z)\frac{1 - z^{-1}}{1 - z^{-1} + z^{-1}F(z)}$$

These are the general expression for a PLL, now as we are working with a second order PLL the loop filter $F(z)$ is defined as:

$$F(z) = K\frac{1 - bz^{-1}}{1 - az^{-1}}$$

By applying the final value theorem leads to:

$$\theta_e[\infty] = \lim_{z\to 1}\{(z-1)\Theta_e(z)\} = \frac{\Delta\omega}{K}\frac{1-a}{1-b}$$

Therefore, for the PLL to be able to correct frequencies, a pole in $a = 1$ its necessary, reducing the expression to:

$$F(z) = K\frac{1 - bz^{-1}}{1 - z^{-1}}$$

Replacing the second order loop filter term in the phase error expression:

$$\Theta_e(z) = \Theta_i(z)\frac{(1-z^{-1})^2}{1+(K-2)z^{-1}+(1-Kb)z^{-2}}$$

If the oscillators have different frequencies at the input, a phase ramp is observed:

$$\theta_i[n] = \Delta\omega\, n\, u[n] \;\leftrightarrow\; \Theta_i(z) = \frac{\Delta\omega z^{-1}}{(1-z^{-1})^2}$$

And the phase error would be:

$$\Theta_e(z) = \frac{\Delta\omega z^{-1}}{1+(K-2)z^{-1}+(1-Kb)z^{-2}}$$

$$\Theta_e(z) = \frac{\Delta\omega z^{-1}}{1-2r\,cos(\omega_o)z^{-1}+r^2z^{-2}}$$

Where r is the modulus and ω_o the phase of the second order pole, and we can define K and b in terms of these values:

$$K-2 = -2r\,cos(\omega_o)$$
$$K = 2(1-r\,cos(\omega_o))$$

$$1-Kb = r^2$$
$$b = \frac{1-r^2}{K}$$

The noise bandwidth of the loop is defined by:

$$B_L = \frac{\omega_n}{2}\left(\zeta+\frac{1}{4\zeta}\right)$$

If we decide to base the PLL according to the values of B_L and damping factor ζ, then we will need to compute the value of ω_n:

$$\omega_n = \frac{2B_L}{\zeta+\frac{1}{4\zeta}}$$

We can compare how the selection of parameters affect the behaviour of the loop by plotting its transfer function with different values, to do so we will use the transfer function in the S domain (Laplace), which is:

$$H(s) = \frac{{\omega_n}^2+2\zeta\omega_n s}{{\omega_n}^2+2\zeta\omega_n s+s^2}$$

With fixed $\zeta = 0.7$ and different values of B_L it can be observed that a narrower bandwidth takes longer to adapt, but it the practice it is observed that with the narrower value, the fluctuation is closer to the reference input, meaning the created sine wave is more spectrally pure. The chosen value for the project is $B_L = 0.02$ as it is an intermediate value and works fine when running the code.

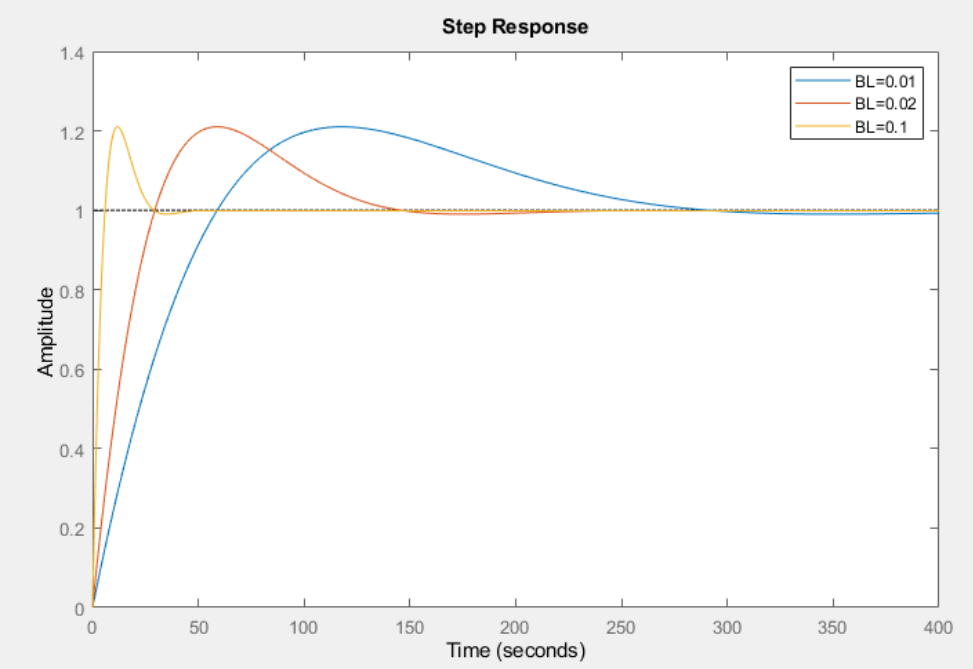


Figure 5.4 Different values of B_L

Now we will plot the transfer function with fixed $B_L = 0.02$ and different values of ζ. The damping factor affects the transient response as explained before in the timing synchronization chapter, here we can see the theory its true, large values of ζ converge faster than smaller values, but as a reminder from the previous chapter, the smaller values have a better ability when tracking the changes in the input signal. A recommended value is 0.7, just as in the Gardner block.

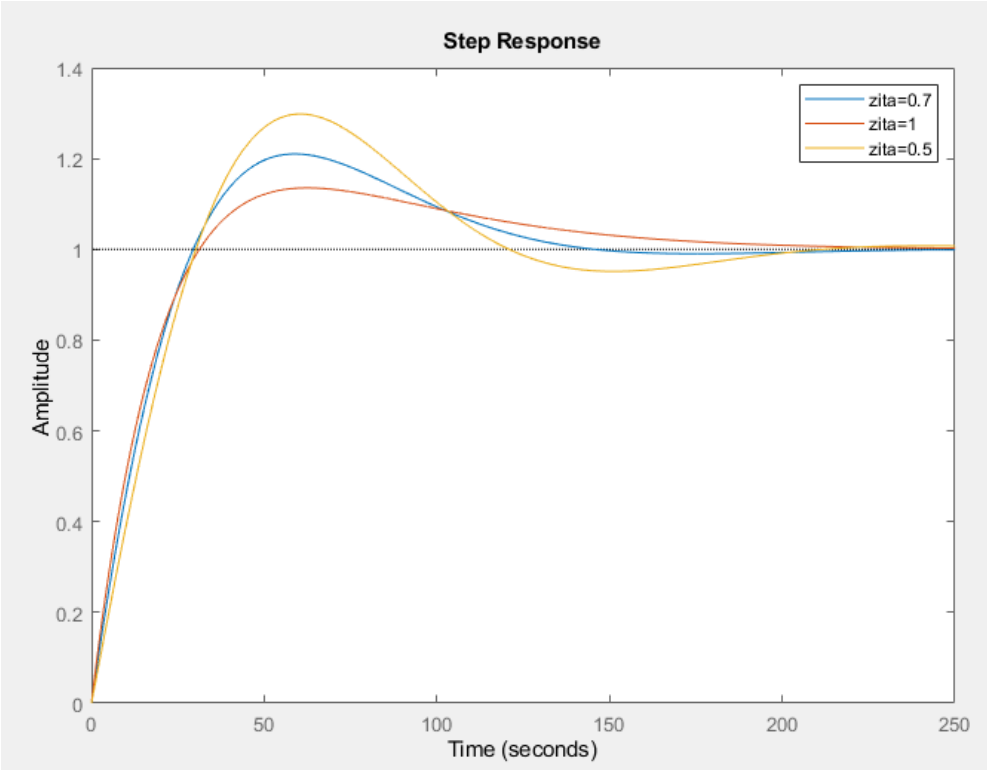


Figure 5.5 Different values of ζ

The code that implements the second order PLL and the Costas Loop error estimator is:

```
P=length(y);
BL=0.02;
zeta=0.7;
wnTs=2*BL/(zeta+1/(zeta*4));
r=exp(-wnTs*zeta);
wo=wnTs*sqrt(1-zeta*zeta);
K=2*(1-r*cos(wo));
b=(1-r*r)/K;
phi=zeros(P+1,1);
s1=zeros(P+1,1);
df=zeros(P+1,1);
e=zeros(P+1,1);
    for n=1:length(y)
        s1(n) = y(n) * exp(-1i*phi(n));
        s2 = sign(real(s1(n)))*imag(s1(n))-sign(imag(s1(n)))*real(s1(n));
        df(n+1)=s2;
        e(n+1) = e(n) + K*df(n+1) - K*b*df(n);
        fi=phi(n) + e(n+1);
```

5.3 MATLAB® SIMULATION

Once the timing synchronization is performed, we can proceed to the carrier recovery stage, so the input of the frequency synchronization is the output of the Gardner block.

To prove the algorithm is working according, we should now see the symbols in a static position, even though we will have a small phase rotation of 22,5° caused by this block which can be easily corrected by multiplying the signal by a complex exponential. This is performed at the output of the equalizer:

```
rotated=salEQ*exp(1i*(pi/8));       %constellation de-rotation
```

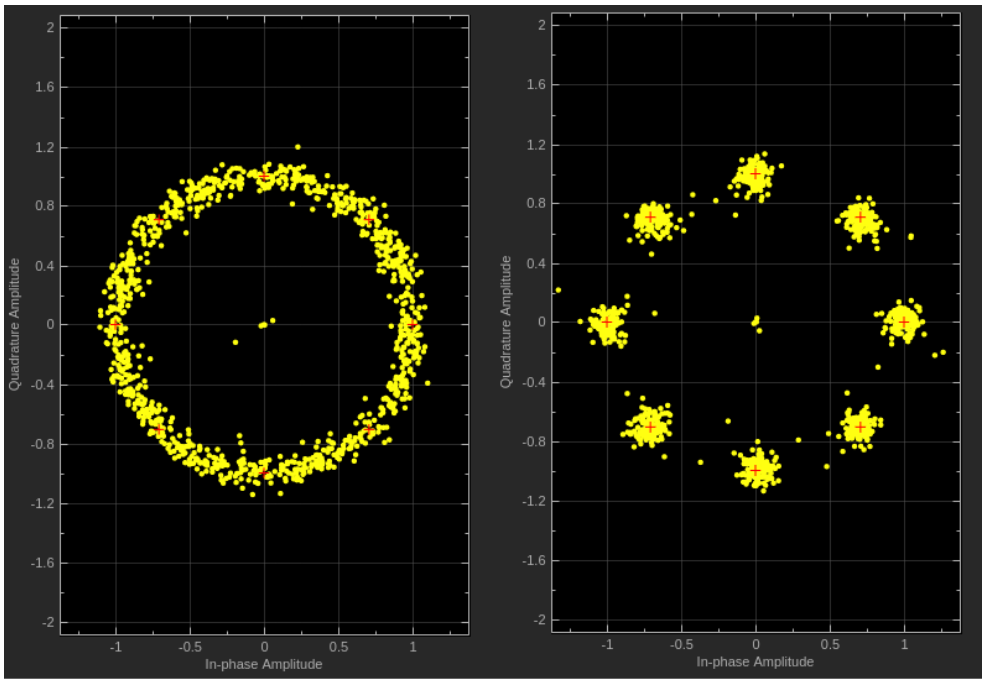


Figure 5.6 Signal before and after frequency synchronization

Now the symbols are in the correct position. The red crosses are the reference constellation set to measure the error vector magnitude (EVM). The constellation may appear like an 8-PSK modulation, but this is just because the rotation of $\frac{\pi}{4}$ performed by the DQPSK is too fast.

By plotting the error signal and the estimated phase, we can see how long it takes for the loop to start working.

In the Figure 5.5 the loop locks approximately after 104 symbols, if we take in count that the symbol rate for TETRA is 18.000 symbols/second (we are now at one sample per symbol), it will be equivalent to 5.8 ms.

The plot of the phase is presented in the Figure 5.6. We can see how it is correctly tracking the phase and now we can use this information to generate a carrier wave with same frequency as the one used in the received signal.

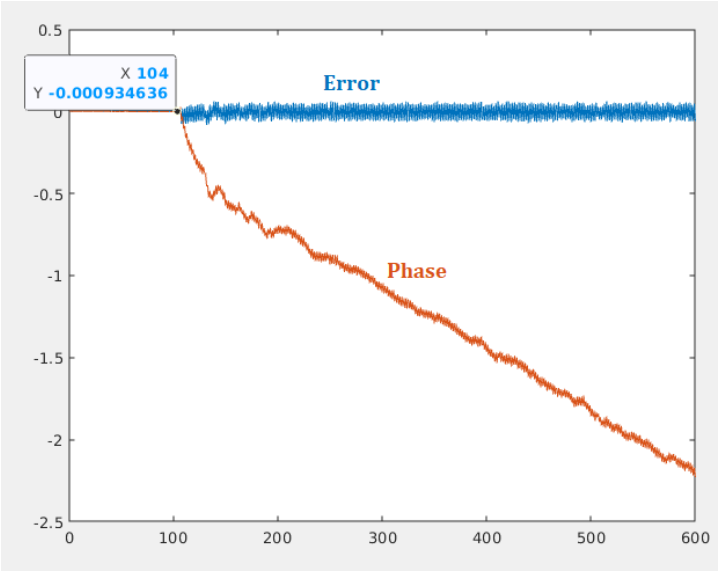


Figure 5.7 Locking time for the Costas loop

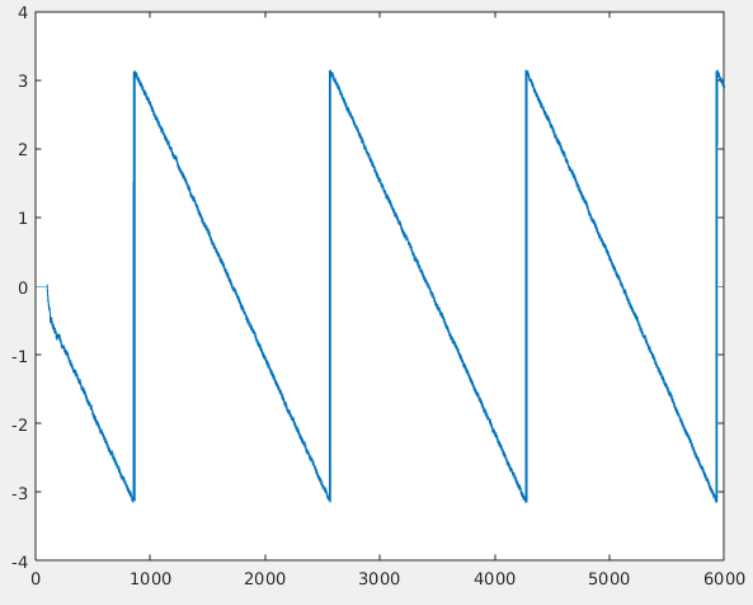


Figure 5.8 Estimated phase after the loop is locked

6. EQUALIZATION

Equalizers are used to mitigate effects of multi-path propagation and Doppler effect, but since the bandwidth of the TETRA signal is small, we will use it here to work as an automatic gain control (AGC) to maintain a suitable signal amplitude.

The equalizer used in this project is based on the LMS algorithm and it is called “blind CMA equalizer”.

A short explanation on how the LMS algorithm works:

The performance function to be minimized is convex and we start at any point of the surface, then take a small step in the direction where the performance function decreases fastest, which is the opposite direction of the gradient of the performance function. By repeating this, the convergence to the bottom of the performance function is assured, where the optimum values that minimize the performance function are found.

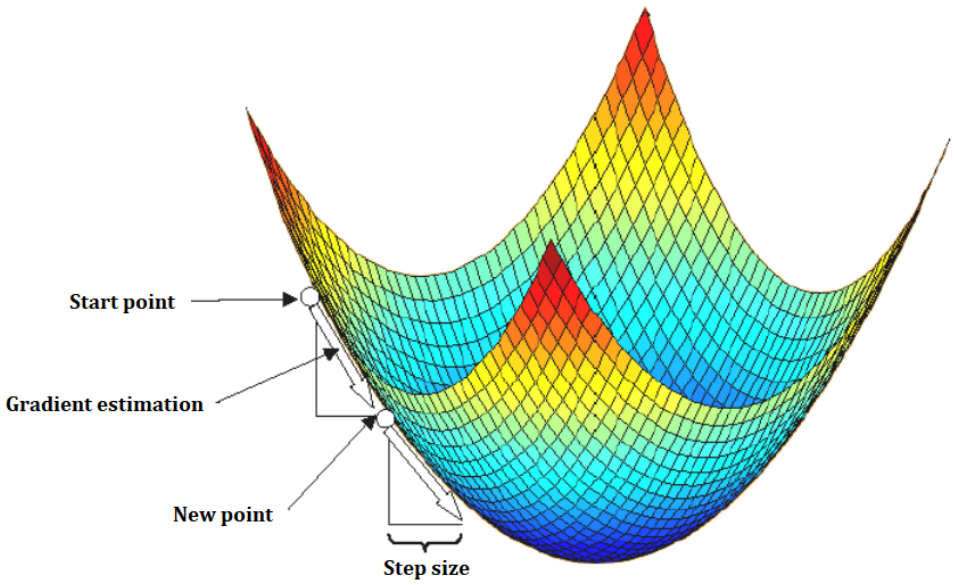


Figure 6.1 Graphical representation of LMS algorithm.

For the case of the blind CMA equalizer, the minimization criteria are:

- The power of binary signals is γ=1.
- We want the power of our signal to be 1, therefore the error signal is: $e[n] = \gamma - |y[n]|^2$

It is necessary to minimize the MSE of:

$$MSE_{CMA} = E\{(\gamma - |y[n]|^2)^2\}$$

So, the LMS algorithm will be:

$$w[k+1] = w[k] - \mu \nabla_w MSE_{CMA}$$

Deriving the MSE_{CMA} we have:

$$w[k+1] = w[k] + 4\mu(\gamma - |y[n]|^2)y[k]x^*[k]$$

Where w are the weights, and its length is equal to the number of the filter order, μ is the step size, x the received signal and y the output of the equalizer.

The filter coefficients can be initialized with a delta in the central coefficient, so the code that implements this function in MATLAB® is:

```
wini=[0 0 1 0 0];    %initial coefficients
PotRef=1;            %reference power
wini=wini(:);        %convert row to column
x=x(:);
mu=0.01;             %initial mu
Lx = length(x);
N=length(wini)-1;    %N=filter order, coeff num.=N+1
y=zeros(Lx,1);
e=zeros(Lx,1);
W=zeros(Lx, N+1);    %coeff. Vector is a row
W(N+1,:)=wini.';     % .' only transpose
for n=N+1:Lx
   xx   = x(n:-1:n-N);  % column vector [x[n] x[n-1] x[n-2]... x[n-N]
   y(n) = W(n,:) * xx;
   Poty = abs(y(n)).^2;
   e(n) = PotRef - Poty;
   W(n+1,:) = W(n,:) + 4*mu * e(n) * y(n) * xx';   % ' transpose and conjugate
 end
salEQ = y;
erEQ = e;
Wfinal = W(end,:);
```

6.1 MATLAB® SIMULATION

The input of the equalizer is the output of the frequency synchronization stage (Costas loop). If we compare both constellation diagrams, we can see the proper behaviour of the equalizer, the values are driven to the optimal symbol amplitude values (red crosses).

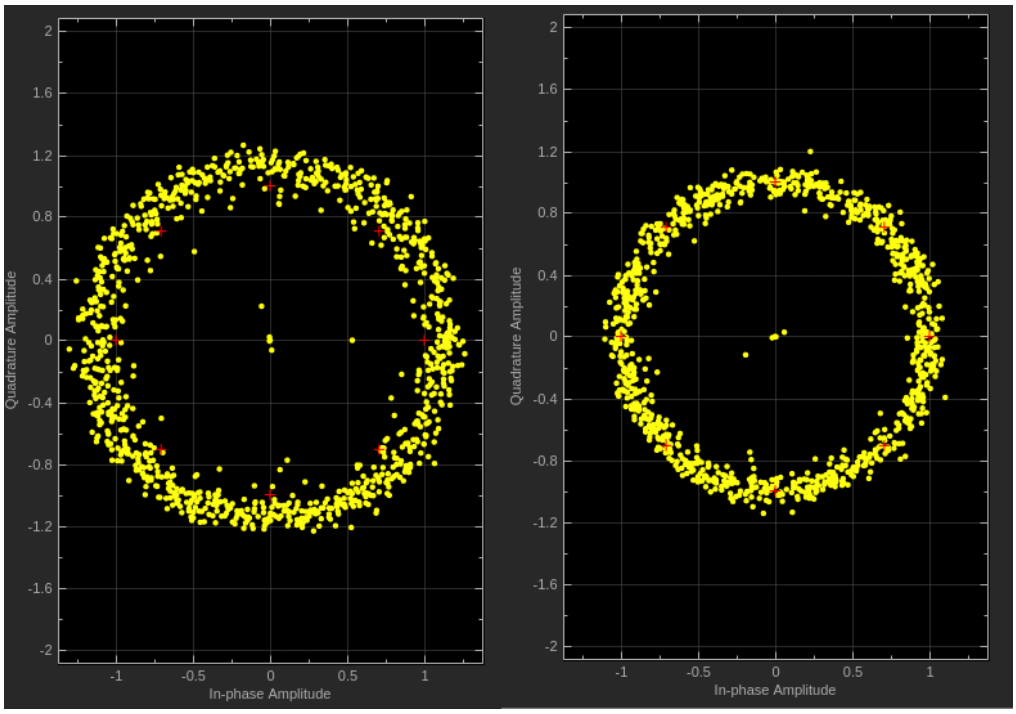


Figure 6.2 Signal before and after equalization

7. DEMODULATION

As explained in the first chapter, TETRA uses a π/4 D-QPSK modulation. It differs from the normal QPSK in the sense that symbols are not defined by a fixed position in the constellation, instead it is defined by the phase difference between two consecutive symbols.

As a comparison, in the Figure 7.1 we have the case of traditional QPSK, where a symbol is linked to a fixed phase. In DQPSK it depends on phase difference between the previous and the actual symbol, so the first symbol will be unknown, and different phases could represent same symbol as in the example.

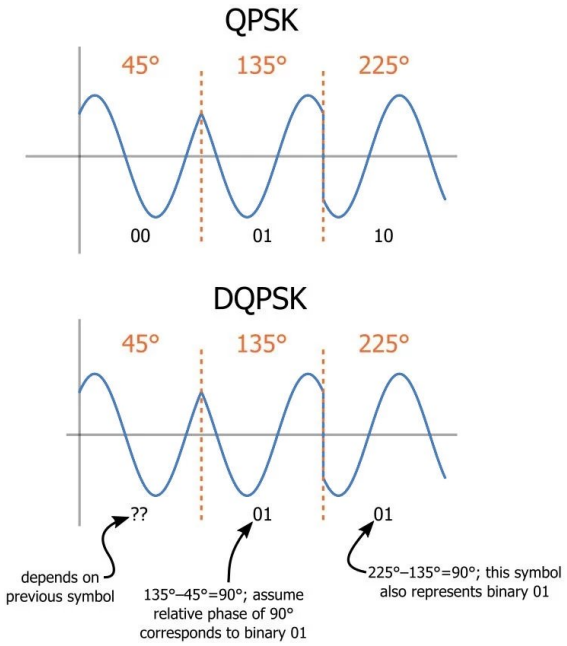


Figure 7.1 Comparison of QPSK and DQPSK symbols

In MATLAB® we can find an object specially designed for this, the `comm.DQPSKDemodulator`. But before we can configure it, we need to understand how the symbols are represented in TETRA according to the standard defined by the ETSI.

The defined symbols and constellation diagram are:

BIT 1	BIT 2	PHASE DIFFERENCE
1	1	-3π/4
0	1	+3π/4
0	0	+π/4
1	0	-π/4

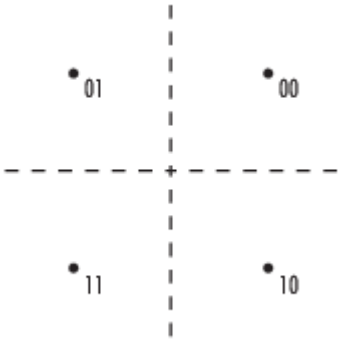


Figure 7.2 Symbol mapping for TETRA DQPSK

We can realize from the symbol mapping that Gray coding is used, which is a relevant parameter for the demodulator object.

Another parameter we need to set is the output data type, which can be binary or symbols, so the selected output will be binary as we need to process the data in bits.

```
%% DQPSK object
Demod = comm.DQPSKDemodulator;
Demod.SymbolMapping='Gray';
Demod.BitOutput=true;
```

7.1 MATLAB® SIMULATION

To test if all the received data is correctly demodulated, we can try to find a frame with the aid of the defined training sequences introduced in the chapter 1. This can be achieved through many ways, one could be for example the cross-correlation function, or one more useful is the command in MATLAB® `findsignal`, which find the defined bits sequence and returns the position where is starts and ends.

After finding the training sequence, we can define where the frame starts according to the frame structure, and check if the content of the different fields is the same as the content defined in the technical documentation.

The sequences used to identify the burst type were introduced in the first chapter. In MATLAB® we will define some variables with these sequences in order to find them in the demodulated bits.

For the Normal continuous and discontinuous downlink burst:

```
NCDB1CH=[1,1,0,1,0,0,0,0,1,1,1,0,1,0,0,1,1,1,0,1,0,0];
NCDB2CH=[0,1,1,1,1,0,1,0,0,1,0,0,0,0,1,1,0,1,1,1,1,0];
```

For the Synchronization continuous and discontinuous downlink burst:

```
SYNC=[1,1,0,0,0,0,0,1,1,0,0,1,1,1,0,0,1,1,1,0,1,0,0,1,1,1,0,0,0,0,0,1,1,0,0,1,1,1]
```

After running the program, the demodulated bits are stored in the variable `bits`, and now we can use the following commands to find the known sequences:

```
[sync_start,sync_stop,dist] = findsignal(bits,SYNC)
[n_start,n_stop,dist] = findsignal(bits,NCDB1CH)
[p_start,p_stop,dist] = findsignal(bits,NCDB2CH)

sync_start = 1379
sync_stop = 1416
dist = 0

n_start =2429
n_stop = 2450
dist = 0

p_start = 389
p_stop = 410
dist = 0
```

The results show the position related to the `bits` array, `dist` tells us about the similarity between the searched and found sequences, where 0 means an exact match.
Now we can determine if the bursts are the type continuous or discontinuous by comparing its content with the presented structures.
Starting with the synchronization burst, both have the frequency correction field, which content is the same but in different bits positions:

1. Synchronization continuous downlink burst

15-94	frequency correction	Bits 1:8 = 1 Bits 9:72 = 0 Bits 73:80 = 1
215-252	synchronization training sequence	1,1,0,0,0,0,0,1,1,0,0,1,1,1,0,0,1,1,1,0,1,0,0,1,1,1,0,0,0,0, 0,1,1,0,0,1,1,1

2. Synchronization discontinuous downlink burst

5-84	frequency correction	Bits 1:8 = 1 Bits 9:72 = 0 Bits 73:80 = 1
205-242	synchronization training sequence	1,1,0,0,0,0,0,1,1,0,0,1,1,1,0,0,1,1,1,0,1,0,0,1,1,1,0,0,0,0, 0,1,1,0,0,1,1,1

We can estimate the starting and ending position based on the returned results. For example, the distance between the frequency correction field and the synchronization training sequence for the Synchronization continuous downlink burst:

$$start = 215 - 15 = 200$$
$$stop = 252 - 94 = 158$$

For the Synchronization discontinuous downlink burst:

$$start = 205 - 5 = 200$$
$$stop = 242 - 84 = 158$$

The distances are the same, therefore we must use another field to identify the type of burst, in the Synchronization continuous downlink burst the first field includes a 12-bit training sequence:

1-12	normal training sequence 3	0,0,0,1,1,0,1,0,1,1,0,1

Now we will have:

$$start = 215 - 1 = 214$$
$$stop = 252 - 12 = 240$$

If we compare the contents based on the previous simulation results:

$$sync\_start - start = 1379 - 214 = 1165$$
$$sync\_stop - stop = 1416 - 240 = 1176$$

If we query these positions, the result is:

```
bits(1165:1176)

ans =     0     0     0     1     1     0     1     0     1     1     0     1
```

So, we have a match, and this is a Synchronization continuous downlink burst and seems like the demodulation is correctly performed.

The same apply to the other types of bursts:

1. Normal continuous downlink burst

1-12	normal training sequence 3	0,0,0,1,1,0,1,0,1,1,0,1
245-266	normal training sequence	1,1,0,1,0,0,0,0,1,1,1,0,1,0,0,1,1,1,0,1,0,0 **OR** 0,1,1,1, 1,0,1,0,0,1,0,0,0,0,1,1,0,1,1,1,1,0

2. Normal discontinuous downlink burst

1-2	normal training sequence 3	0,1
235-256	normal training sequence	1,1,0,1,0,0,0,0,1,1,1,0,1,0,0,1,1,1,0,1,0,0 **OR** 0,1,1,1, 1,0,1,0,0,1,0,0,0,0,1,1,0,1,1,1,1,0

For the Normal continuous downlink burst:

$$start = 245 - 1 = 244$$
$$stop = 266 - 12 = 254$$

And the positions related to the received bits:

$$n\_start - start = 2429 - 244 = 2185$$
$$n\_stop - stop = 2450 - 254 = 2196$$

```
bits(2185:2196)

ans =   0     0     0     1     1     0     1     0     1     1     0     1
```

We have a match, so it is a Normal continuous downlink burst, we can double check with the tail bits sequence:

501-510	normal training sequence 3	1,0,1,1,0,1,1,1,0,0

$$start = 501 - 245 = 256$$
$$stop = 510 - 266 = 244$$

$$n\_start + start = 2429 + 256 = 2685$$
$$n\_stop + stop = 2450 + 244 = 2694$$

```
bits(2685:2694)

ans =     1     0     1     1     0     1     1     1     0     0
```

From the results we can see the data is correctly demodulated, so the all the receiver blocks are working as expected.

8. BIBLOGRAPHY

[1] Farhang-Boroujeny, Behrouz. Signal Processing Techniques for Software Radios. ECE Department, University of Utah. 2010

[2] Stewart, Robert W. Software Defined Radio using MATLAB® & Simulink® and the RTL-SDR. Strathclyde Academic Media. 2015

[3] Rice, Michael. Digital Communications: A Discrete-Time Approach. Pearson Prentice Hall. 2009

[4] ETSI TS 100 392-2. Terrestrial Trunked Radio (TETRA); Voice plus Data (V+D); Part 2: Air Interface (AI). 2020

[5] Collins, Travis F. Software-Defined Radio for Engineers. Artech House. 2018

[6] https://en.wikipedia.org/wiki/Cascaded_integrator%E2%80%93comb_filter

[7] https://www.soundonsound.com/sound-advice/q-what-exactly-comb-filtering

[8] https://www.dsprelated.com/showarticle/1337.php

[9] https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-modulation/digital-phase-modulation-bpsk-qpsk-dqpsk/

[10] https://zipcpu.com/dsp/2018/03/30/quadratic.html

Declaration

I declare that I have written this thesis independently. All passages taken literally or in essence from published or unpublished works by others or the author himself/herself have been marked as taken. All sources and aids that I have used for the work are indicated. The thesis has not been submitted to any other examination authority with the same content or in essential parts.

Cologne, 27.07.2022

Signature

TH Köln- Campus Deutz
Betzdorfer Str. 2
50679 Köln
www.th-koeln.de

Technology
Arts Sciences
TH Köln