VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

ACARS
Aircraft Communications Addressing and Reporting System

2010 Jaroslav Henner

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. 7. 2010 ....

Rád bych poděkoval rodičům za to, že mi pomohli k tomu, že si zde tuto řádku mohou přečíst.

Abstrakt

Tato práce popisuje návrh efektivního koherentního jednoprůchodového demodulátoru signálu ACARS, implementovaného s využitím toolkitu GNU-Radio. Jsou použity Costasovy fázové závěsy, Viterbiho algoritmus a obnovu taktu pomocí polyfázní filtr-banky. Také je zde popsán způsob odstranění fázové nejednoznačnosti MSK signálu. Implementovaný demodulátor je schopen demodulavat v reálném čase, se čtením dat ze zvukové karty počítače, ale také může pracovat off-line, se čtením dat ze standardního vstupu nebo ze souboru. Obojí s velmi dobrým počtem korektně demodulovaných zpráv.

Klíčová slova: ACARS, demodulátor, MSK, DSP, zpracování digitálního signálu, signál

Abstract

This work describes the design of an effective coherent signle-pass demodulator for ACARS, implemented with the GNU Radio toolkit, using two Costas loops, Viterbi algorithm and a polyphase filter-bank clock recovery. A way of removing the MSK signal phase ambiguity is also described here. Implemented demodulator is capable to demodulate in real-time, reading the data from a computer sound card, or it can work off-line, reading the data from standard input or file, both with very good number of correctly demodulated messages.

Keywords: ACARS, demodulator, MSK, DSP, digital signal processing, signal

Used abbreviations and acronyms dictionary.

µP – Microprocessor
ACARS – Aircraft communications addressing and reporting system.
AM – Amplitude modulation.
AWGN – Additive white gausian noise.
Baseband – Frequency rage around zero frequency.
CL – Costas loop.
DSB+C – Double-side band plus carrier. Modulation scheme.
DSP – Digital signal processing.
EOF – End of file.
FFSK – Fast frequency-shift keying.
FSM – Finite state machine.
FST – Finite state transducer.
GR – GNU Radio
LSB – Least significant bit.
Passband – Frequency range around carrier frequency.
PLL – Phase locked loop.
MSB – Most significant bit.
MSK – Minimum-shift keying.
NRZ – Not return to zero.

Contents

List of Tables

List of Figures

1 Introduction.

When studied the Computer Science and Technology on VSB-TUO, I took a subject about digital signal processing. I started to like this field and I wanted to study it more. After finishing this course, it was the time to choose the topic of final thesis. There were some interesting topics offered, but there was also this topic — designing of a software demodulator for ACARS, which I liked the most.

After subscribing for this topic, I started to gather some information about the ACARS. There is not much of free protocol descriptions on the Internet. I managed to find some web pages written by some radio amateurs that briefly described the protocol. I begun analyzing a recorded messages in sound editor, searching for some pattern. I found some pattern in the begining. I also found which wave means which bit, because the start of message that I demodulated by hand was similar to the text that I have been given with the recording.

Then I found a great information source — the unpublished draft of ARNIC's specification 618 [ARNIC, 2008]. It really helped me with further development of my demodulator, but it was also quite disappointing, because I was proud of me to puzzle the keying scheme out.

Very very soon after my subscribing the thesis topic I was able to correctly demodulate some of the messages in Python, using NumPy and SciPy packages, using a frequency discrimination.

Then, I read some articles about a Costas loops. I wrote some code in Python. The results of the algorithm with CLs were promissing, but the code was horribly slow, because the previous approach extensively used a vector processing provided by NumPy. The CLs couldn't be implemented using the vector processing, because of the feedback in the PLL — the computation of every sample going out of the CL depends, among other things, on the previous sample values.

After some time, I was advised to use a GNU-Radio, which allowed me to lower the CPU processing demand. GNU Radio is a toolkit that provides many signal-processing blocks written in C++, even the Costas loops. These blocks can be connected together using Python. Interfacing between the native code and python code is done using SWIG. GNU Radio is simple to use, versatile and fast enough. Any block can run in separate thread, thus parallel processing is supported. It also contains a filter design tools. Although the documentation sometimes lacks some important information, the source code is very easy to read and so one can grasp the missing for himself.

But there (still) was one big problem, that I couldn't solve for very long time — how can I keep the bits in sync? Even the non-coherent demodulator lacked (and needed) some synchronisation of bits. No one can expect the sound card to sample the signal so that every time we would advanced by n samples, we were dealing with next bit. None of my demodulators could deal with the samples going out of sync with the bits. That was a big problem that I couldn't overcome, until I tried the Mueller-Muller and polyphase clock recovery blocks in GNU Radio. They worked well.

When reading about the MSK, I realized that the Viterbi algorithm can be used to make the demodulator's performance even better. I had to experiment a while to find a

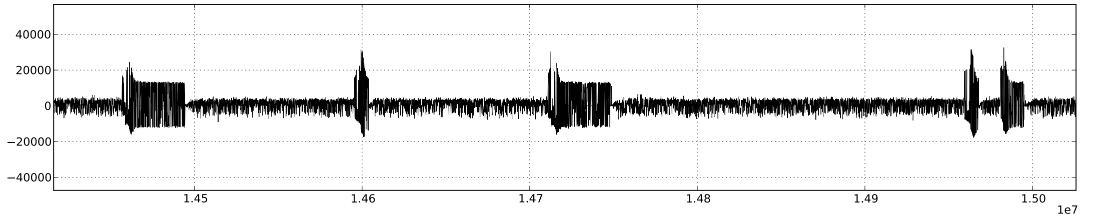


Figure **1:** Five ACARS messages.

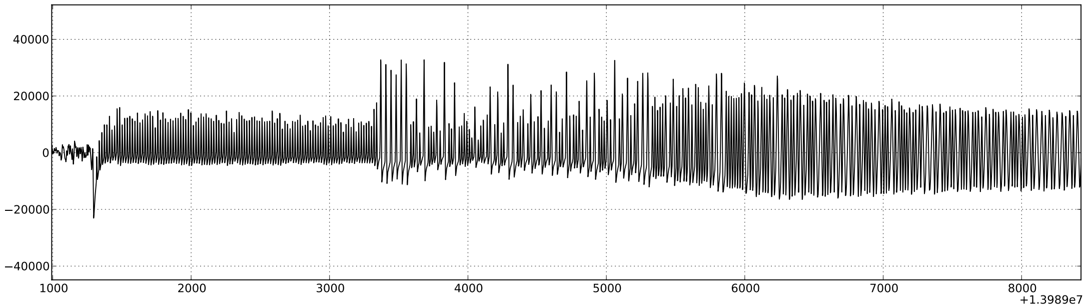


Figure **2:** The message detail.

proper setting, but when I got it working, the count of correctly demodulated messages doubled! Then, I was satisfied with the results of the demodulator because it demodulated almost all of the messages I could see in the records, so I started to think about the back-end of my demodulator.

After finding out how can I pass the data from GNU-Radio to Python (a file descriptor and pipe can be used), I wrote the final message processing and parsing.

This was the introduction and history of this thesis. Now comes the section about the history of ACARS.

2 ACARS History.

Airline Communications, Addressing and Reporting System (ACARS) is a data messages delivery system used for communication between the airline, the aircraft and air traffic control (ATC) station.

Prior the ACARS system, all communication between the ground and the aircraft had to be done using voice transmitted by very high frequency (VHF) or high frequency (HF) radios. The voice system has couple drawbacks — the voice messages are not easy to automatically generate and analyze, so the human resources had to be used to process them. This increases the costs and it also increases the workload of the airliner crew. Because the air is a shared communication medium, and the voice messages occupies too much of bandwidth and time, the voice system doesn't allow many aircrafts to communicate.

In the 1978, the voice-transmission VHF network was utilized also with ACARS. ACARS is a system for transmission of data messages. Instead of the human voice, the modulated signal carrying the data is transmitted over VHF similarly to the modems that were used to connect to the Internet over the fixed telephone lines. [Various authors, 2002] [Wikipedia, 2010]

Among the first kinds of messages delivered by ACARS were so-called OOOI events — Out of the gate, Off the ground, On the ground, and Into the gate, generated by processing of the information from several aircraft sensors and controls, like door-opening sensor and brakes status.

The system was evolving and expanding with time. The number of message types, ground stations, transmitted messages and airliners grew:

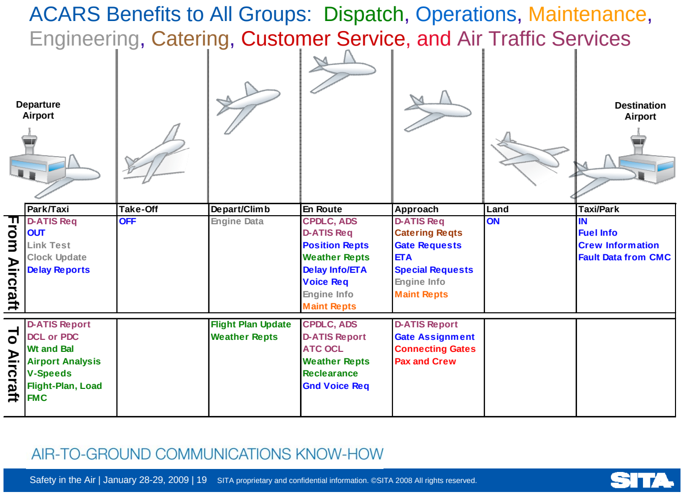
> At its peak, the ACARS network which now includes satellite and High Frequency Data Link (HFDL) air/ground subnetworks, carried 22 million messages in one month. More than six thousand aircraft are equipped with ACARS avionics.[Oishi, 2002]

Today, ACARS supports a maintenance messages, delivering information about the aircraft device failures and operation characteristics, and also aircraft operation conditions and weather reports, in real-time, allowing better planning of ground maintenance actions. The system is utilized even also for transmissions of flight management information — uploading the flight plans to the aircraft. Such messages are received by the ACARS unit and then forwarded to Flight Management Unit. This allows the flight plan to be updated during the flight.

Besides all of these structured messages, the ACARS is also capable to deliver messages without any predefined structure, allowing the crew or the ground to send human-written text whenever it is needed.

Today, the ACARS becomes obsoleted by Aeronautical Telecommunication Network which uses ISO/OSI standards and protocols.

[Various authors, 2002][Wikipedia, 2010][ARNIC, 2008]



ACARS Benefits to All Groups: Dispatch, Operations, Maintenance, Engineering, Catering, Customer Service, and Air Traffic Services

Departure Airport
Destination Airport

	Park/Taxi	Take-Off	Depart/Climb	En Route	Approach	Land	Taxi/Park
From Aircraft	D-ATIS Req OUT Link Test Clock Update Delay Reports	OFF	Engine Data	CPDLC, ADS D-ATIS Req Position Repts Weather Repts Delay Info/ETA Voice Req Engine Info Maint Repts	D-ATIS Req Catering Reqts Gate Requests ETA Special Requests Engine Info Maint Repts	ON	IN Fuel Info Crew Information Fault Data from CMC
To Aircraft	D-ATIS Report DCL or PDC Wt and Bal Airport Analysis V-Speeds Flight-Plan, Load FMC		Flight Plan Update Weather Repts	CPDLC, ADS D-ATIS Report ATC OCL Weather Repts Reclearance Gnd Voice Req	D-ATIS Report Gate Assignment Connecting Gates Pax and Crew		

AIR-TO-GROUND COMMUNICATIONS KNOW-HOW

Safety in the Air | January 28-29, 2009 | 19 SITA proprietary and confidential information. ©SITA 2008 All rights reserved.

SITA

Figure **3:** The various kinds of ACARS messages sent in various flight phases. Image from [Mattos, 2009]

2.1 ACARS network.

The VHF signal travels on near-line-of-sight trajectory from the transmitter and bends over the horizon and hills. Comparing to other kinds of terrestrial VHF transmissions, like TV broadcasting, the range of ACARS signal transmissions is relatively long, because the airliners cruising altitudes are about 10km — compare that with the altitudes of the VHF terrestrial TV broadcasting antennas.

2.2 Software demodulation.

An ordinary VHF voice system with amplitude modulation was used as a channel for the ACARS data transmissions. Therefore an ordinary VHF Amateur radio receiver can be

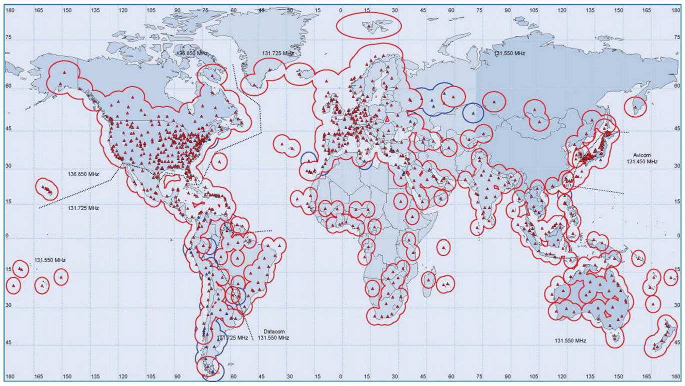


Figure **4:** The SITA VHF Remote Ground Stations coverage. Map as of March 2008, altitude 30,000 feet, on-line RGS are in red, planned are in blue. [Mattos, 2009] used to demodulate the ACARS messages from the radio frequencies to the acoustic range. Then, an ordinary computer sound card can be used to digitize the acoustic range signal. A second demodulation can be performed in the computer using a software demodulator. Design of such demodulator is the topic of this thesis.

3 Theory.

In this section you can find a theory that is needed for the most of the thesis.

3.1 Signal.

Signal is the output of a system — a function of some physical quantity. Quite often a function of time or frequency. The values in one time instant may be discrete or continuous — value can be Real or integer number. The time domain of the signal may also be discrete (sampled) or continuous. Signals that we want to process on digital computer must be discretized in time (sampled) — this can be done almost losslessly if the signal which we want to process is band-limited (the Nyquist-Shannon sampling theorem). Then it also must be quantized — discretized in values because the count of numbers the computers work with is limited. [1]

In this thesis, I will be talking mainly about discrete-valued, discrete-time signals, because I'm dealing with the software demodulator.

3.2 Linear, time invariant system (LTIS).

The linearity and time invariance are a very useful properties of systems. LTIS are easy to analyze. They can be combined together or split apart easily. We can also swap two LTIS in series while the result remains unchanged.[Proakis and Manolakis, 2006]

Sometimes the term "shift invariant" is used in spite of "time invariant," but it means the same. Smith gives nice definition, so I present it (with the references to some pictures removed):

> A system is called linear if it has two mathematical properties: homogeneity (hōma-gen-ā-ity) and additivity. If you can show that a system has both properties, then you have proven that the system is linear. Likewise, if you can show that a system doesn't have one or both properties, you have proven that it isn't linear. A third property, shift invariance, is not a strict requirement for linearity, but it is a mandatory property for most DSP techniques. When you see the term linear system used in DSP, you should assume it includes shift invariance unless you have reason to believe otherwise. These three properties form the mathematics of how linear system theory is defined and used.
>
> ...
>
> Homogeneity means that a change in the input signal's amplitude results in a corresponding change in the output signal's amplitude. In mathematical terms, if an input signal of $x[n]$ results in an output signal of $y[n]$, an input of $kx[n]$ results in an output of $ky[n]$, for any input signal and constant, k.
>
> The property of additivity ... Consider a system where an input of $x_1[n]$ produces an output of $y_1[n]$. Further suppose that a different input, $x_2[n]$,

---

[1] The Nyq.-Sha. sampling needs an infinite sequence of samples to fully reconstruct the band limited signal, but the error induced by using only finite sequences can be limited to be acceptably small, therefore the sampling can be done almost losslessly.

> produces another output, $y_2[n]$. The system is said to be additive, if an input of $x_1[n] + x_2[n]$ results in an output of $y_1[n] + y_2[n]$, for all possible input signals. In words, signals added at the input produce signals that are added at the output.
>
> ...
>
> Shift invariance means that a shift in the input signal will result in nothing more than an identical shift in the output signal. In more formal terms, if an input signal of $x[n]$ results in an output of $y[n]$, an input signal of $x[n + s]$ results in an output of $y[n + s]$, for any input signal and any constant, s. Pay particular notice to how the mathematics of this shift is written, it will be used in upcoming chapters. By adding a constant, s, to the independent variable, n, the waveform can be advanced or retarded in the horizontal direction. For example, when $s = 2$, the signal is shifted left by two samples; when $s = -2$, the signal is shifted right by two samples.[Smith, 1998]

The signals can be expressed as a sum of sinusoids with various frequencies and phases. From the additivity principle, the linear system cannot produce output with a nonzero amplitude in frequencies that are of zero amplitude in the input signal. It can only amplify or attenuate or change the phase of the frequencies that are already nonzero in the input.

3.3 Communication channel.

Communication channel can be viewed as a system connecting the information source with information sink. Ideal channel might be seen (modeled) as a system that whatever signal is passed in, the same signal is retrieved on the output. But this is a very ideal case. In real world, there are limits. Signal on the output of the system may be attenuated significantly. It can be distorted, which means that the shape of the signal waveform is alternated somehow. Such distortions can be: a harmonic distortion caused by some nonlinearity in system that the signal is passing through; phase distortion caused by nonlinear phase response of channel; some frequencies may be attenuated more then other frequencies — caused by non-constant frequency response. The signal on the output of the channel is often time-delayed, and also have some noise added. All of this can cause a mistakes in reception of the data, or the reception can be so erroneous that it is impossible to transmit any information.

There are many models describing the channels. Many wired channels can be seen as a linear, time-invariant systems (LTIS) with additive white Gaussian noise (AWGN). Radio waves channels may vary in time for example due to effects called fadings. For example the change of amount of water vapors in troposphere or ionization of ionosphere may change the attenuation of the channel. Thus the radio channels are often modeled as linear, time-variant, AWGN channels. The time-variances cannot be canceled by LTIS. The time variant systems, such as adaptive filters, must be used to do that.

When we want to pass a signal through a channel we often have to transform the signal into some form that can be passed through the channel easily, and then, on the output of the channel, we change it back to the original form as it is possible. That means that we use some another systems, transmitters and receivers, to change the signal into the form that can be passed through the channel. The system: a transmitter connected to a channel connected to a receiver; can be viewed as a new communication channel.

3.4 Modulation, keying.

The process of transforming the signal into another form is called modulation or keying. ”Modulation” is often used when talking about transforming a continuous-time signal into continuous-time signal, and the word ”keying” is used when talking about transforming a discrete-time signal into continuous-time signal. These terms are often vague-defined and interchanged, but this doesn’t matter much because it still is the method of changing the signal into some form that can pass through a channel.

In modern computers, the information is encoded as a sequence of random variables carrying an information — bits. Transmission of the two-valued variables are often not suitable for the transmission system — the transmission system can often perform better when more information is transmitted in a time instant. The message is split into sequence of bits which are then encoded to *symbols*. The modulator then transmits the symbols one by one. In Phase-Shift Keying, the set of symbols is a set of some number of periods of phase shifted carrier sinusoids. In QuadriPhase-Shift Keying (QPSK), a set of four 90° shifted sinusoids is used for the *symbols*.

Remark 3.1 People use a huge amount of distinct letters for writing. Imagine how long the books would be if people used only two letters.

Remark 3.2 People use modulation really often. Maybe so often that they don’t even notice. When some person have an idea to share — information, he can use the language and vocal tract to encode the idea into sound waves, that the other people can hear with their ears and interpret it in the brain. That person — transmitter must speak slowly, loudly, and pronounce as well as it is enough to allow the hearer, the receiver, to acceptably understand the message. The hearer may sometimes misinterpret those parts of the messages, that were changed by distortion or noise — receiver can make a mistakes. Many times the mistakes can be detected and sometimes even corrected using grammar rules of the language for example, which can be viewed as an analogy to the error detecting and correcting in communication technology.

3.5 Constellation diagram.

Constellation diagram is a scatter plot of symbols integrated by one symbol time length and sampled. It is a geometric representation of the keying scheme and it is an important tool for analyzing the keying schemes.

If a modulation symbols can be expressed as a Complex numbers of the form

$$S = Ee^{\mathbf{i}\phi} = E\cos\phi + \mathbf{i}E\sin\phi$$

where the energy of the symbol is E and the phase of sinusoid is ϕ, we can draw a point on the Complex plane which represents the symbol.

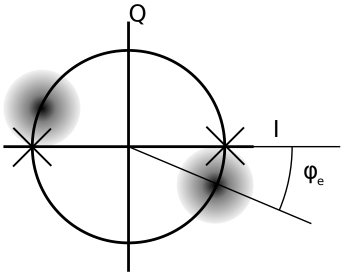


Figure **5:** Constellation diagram of binary PSK. The two crosses are the constellation points, the two clouds is the distribution of the symbols. The angular difference between the cloud and the cross is the phase error.

The demodulators are often composed of several signal-flow branches, very often only two branches are needed. One detects the energy in Real part of the signal — so called I channel, the second detects the energy in Imaginary part of the signal — the Q channel. The Figure 8 shows a Costas loop with clearly recognizable Quadrature demodulator which contains the I and Q channel branch. The presence of energy in each channel then can be mapped to the modulation symbols.

The constellation diagram shows the distribution of the energy over the signal-space. The energy in I channel can be plotted on the Real axis, and the energy in the Q channel is on the Imaginary axis. When we plot the constellation diagram of received signal, the points are spread in so-called cloud around it's ideal position. The shape and position of the cloud can indicate some phenomenas, like: noise, phase jitter of reference frequencies, inaccuracies and nonlinearities and so on.

3.6 Costas loop.

For coherent (phase aware) demodulation, the demodulator's carrier frequency reference must be phase-synchronized with the carrier of the signal being received. Some keying schemes includes the carrier reference in the transmitted signal, but because of the spectrum and energy efficiency reasons, the carrier is often not present in the spectrum of the transmitted signal. This is the case of the QPSK and MSK. But for coherent demodulation, it is essential to have some reference for generating the carrier, which will be than mixed with the received signal. [2]

The Costas loop is a negative-feedback control system used for the carrier recovery and for taking the signal into the baseband. As any negative-feedback system, it estimates an error between the input signal and some reference signal and makes corrections to minimize that error.

---

[2] There is also a possibility to use worse-performing non-coherent demodulation.

3.6.1 Principle of CL.

Suppose we have taken our signal on the carrier f_c to the baseband by mixing it with the carrier reference f_o and low-passing the result. If no effort on synchronization was made, the difference between the received signal carrier f_c and the reference f_o will cause the distribution of received symbols to rotate. The frequency of this rotation is $f_c - f_o$ (a frequency error). This rotation would cause an erroneously demodulated data, so some corrections to the mixing frequency signal must be made.

Now suppose, that the reference frequency is same as the frequency of the carrier. Then the symbols distribution doesn't rotate — it is steady, but it still can be rotated by some angle — a phase error ϕ, as shown on Figure 5. The phase error is an difference between the carrier of the received signal and the reference frequency. The phase error can also lead to bad performance of coherent detector. [3]

Both kinds of errors can be minimized using the Costas loop.

Figure 6 shows a one variant of the Costas loop. The Complex signal input, on the figure denoted with $\tilde{x}_n$, is multiplied with the correcting signal $\exp(-j\theta)$. Note that $e^{\mathbf{i}x}e^{\mathbf{i}y} = e^{\mathbf{i}(x+y)}$. From the multiplier the signal goes to the Detector which decides which symbol is being received and puts it on the output — $\hat{a}_n$. Error generator estimates the phase error between the decided symbol $\hat{a}_n$ and the corrected symbol, creating an error signal $e[n]$, which is multiplied by a constant factor γ controlling the loop gain. The error signal is then filtered in Loop filter, which acts as an integrator. The "exp(·)" block generates the correcting signal, so the loop is closed.

Note that the "exp(·)" block and the loop filter acts like a Voltage Controlled Oscillator (VCO).

If there is a phase error, the filter will keep increasing it's output as long as there is a positive value on it's input. After some number of samples, the phase error will get minimized to negligible value (it will be zero after infinite time). The output of the loop filter will be a some measure of the average phase error.

Remark 3.3 This is a principle of PI regulator from Control Theory.

Our loop will minimize the effect of the phase error between the carrier and reference frequency, but if there is a frequency error, the loop will stop the rotation of the symbols distribution, but will not minimize the error completely. A frequency is a derivative of a phase, so if there is constant nonzero frequency error, then there is a constant change in the phase error, so the output of the filter — an averaged (damped) error will always drift some constant behind the real error.

This can be solved by augmenting the loop filter by one more integrator which will compensate the residual error. This way we created a second-order control system. The augmented filter is on Figure 7. [Haykin, 2001]

In some designs, the Detector performs only a identity function (the detector is not present in the loop), so the Error generator would be given two same streams. Thus it must create an error only using the statistical parameters of wanted constellation. In this

---

[3] The non-coherent detector doesn't mind the phase error.

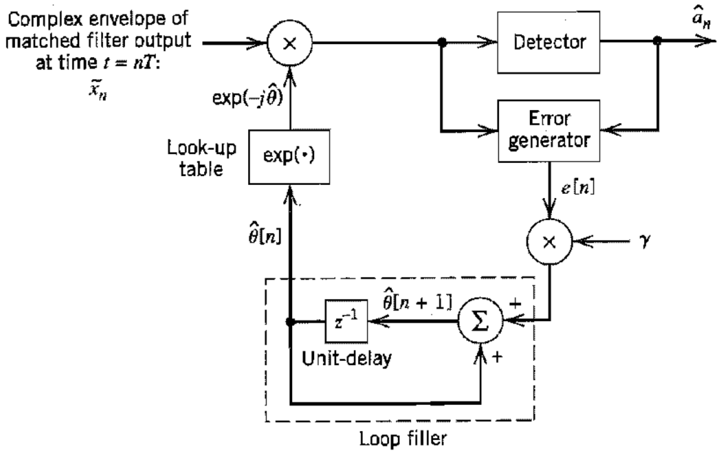


Figure **6:** Schema of Costas loop. [Haykin, 2001]

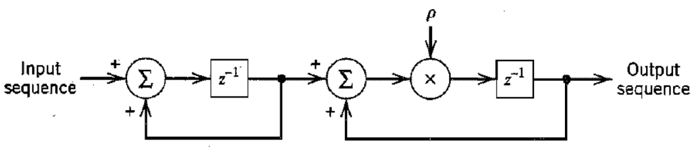


Figure **7:** Schema of second order filter for Costas loop. [Haykin, 2001]

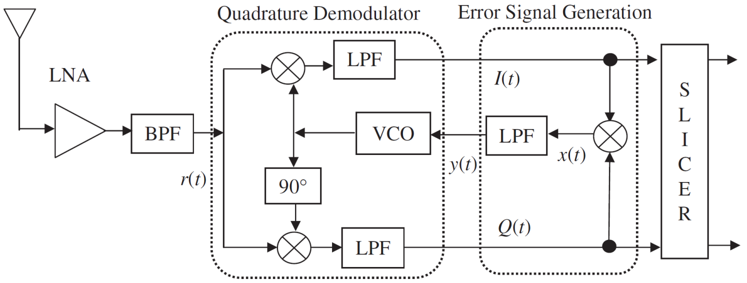


Figure **8:** Schema of Costas loop using $I \cdot Q$ as an error signal.[Boccuzzi, 2008]

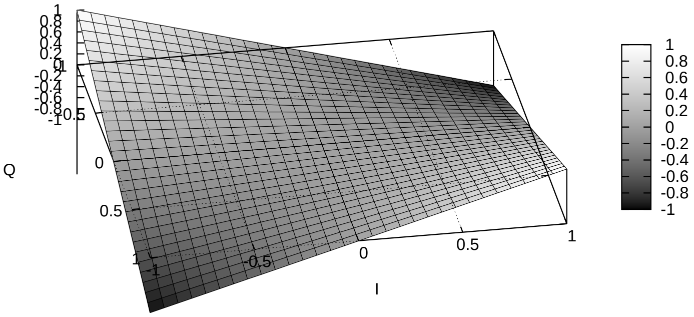


Figure **9:** Error signal function, $e(I, Q) = I \cdot Q$.
Be careful about which axis is which line. The position of axis labels can be misleading. The point in middle of the bottom line has an ordinates: $I = 0$, $Q = -1$. The point in left bottom is $I = -1$, $Q = 1$. I am sorry about the text overlapping.

case, the product of I and Q channel is used as an approximation of a phase error. The function is shown on Figure 9.

One could think that this function has four angles, where the error is zero, thus such Costas loop will have four points in constellation that it can be locked-on. But two of these angles — the Q axis line, are unstable. They are the lines, where the loop is equally likely to approach the lock in any direction. The Costas loop on Figure 8 is a circuit using such error function.

To imagine how the loop acquires the lock, I present a phase-plane portrait of phase-locked loop on Figure 10. The PLL is often used in communications. It is similar to the Costas loop, but it uses only one point in the constellation to lock on.

3.6.2 Output of Costas loop.

To analyze the output of the CL, suppose that it is locked on frequency f_h. Costas loop can be viewed as a coherent receiver and the carrier phase synchronizer for some kind of PSK in one block. The coherent receiver performs the Complex multiplication of the input signal $A(t)e^{\mathbf{i}2\pi ft+\mathbf{i}\phi(t)}$ with the signal from oscillator $e^{-\mathbf{i}2\pi f_o t-\mathbf{i}\phi_o}$ tuned to frequency f_o with phase ϕ_o. $A(t)$ is the amplitude of the input signal in time t, f is the carrier frequency of the input signal and the $\phi(t)$ is the instantaneous phase of the input signal in time t. Therefore we can say, that output of a Costas loop with oscillator in such state will be

$$A(t)e^{\mathbf{i}2\pi ft+\mathbf{i}\phi(t)} \cdot e^{-\mathbf{i}2\pi f_o t-\mathbf{i}\phi_o} = A(t)e^{\mathbf{i}2\pi(f-f_o)t+\mathbf{i}(\phi(t)-\phi_o)}$$

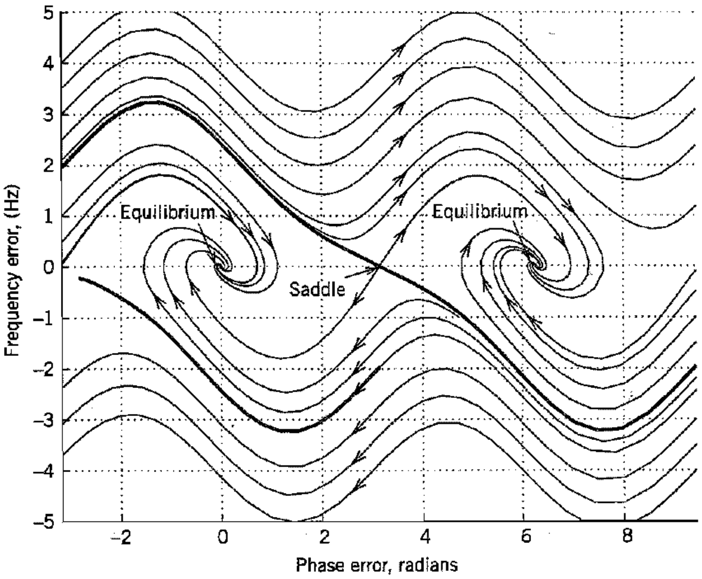


Figure **10:** Phase-locked loop phase-plane plot. Critical damping and sinusoidal modulation [Haykin, 2001].

Remark 3.4 Simply, the Costas loop changes the frequency of rotation with center in the 0 of the point in the Complex plane.

4 ACARS transmission system.

ACARS message bits are keyed using Minimum-shift keying into acoustic frequency range signal which is then used to modulate the radio frequency carrier of about 130MHz using the amplitude modulation.

This thesis deals only with that part of whole transmission chain, that demodulates the message bits from the acoustic range signal received by the VHF receiver — in other words, only the Minimum-shift keying (MSK) demodulator have to be described here. Before constructing such demodulator, the keying scheme should be described.

4.1 Keying scheme.

In ACARS, two instantaneous frequencies $f_l = 1200 \pm 0.02\%\text{Hz}$ and $f_h = 2400 \pm 0.02\%\text{Hz}$ represents the symbols keyed into the acoustic range. The frequency f_h represents a no change in bit value, respectively to previous bit. A f_l represents a bit with inverted value of previous bit being transmitted. The used symbol transmission speed is $2400 \pm 0.02\%$ baud so the symbol duration is

$$\tau = \frac{1}{2400} \approx 416.7\mu\text{s}.$$

Note that one symbol duration is one half-period of f_l or whole period of f_h. Instantaneous frequencies should be changed exactly in the zero-crossings of transmitted signal. This means that when the frequencies are changed, their phases should be equal, so the phase of transmitted signal is continuous, but not smooth, function of time. The Figure 12 shows the waveform of ACARS signal.[ARNIC, 2008]

In other words, a subclass of Frequency-shift keying (FSK), the minimum-shift keying (MSK), is used. The frequency deviation ratio (aka. modulation index) of MSK is

$$h = (f_h - f_l) \cdot \tau = (2400 - 1200)\frac{1}{2400} = \frac{1}{2}.$$

This scheme is also known as Fast Frequency-shift keying (FFSK) because of the space between the frequencies used for signaling is only a half of conventional FSK, thus the frequency changes happens at least two times more often than in FSK. It can also be viewed as variation of OQPSK.[Pasupathy, 1979]

4.1.1 Minimum shift keying (MSK).

When expressed in the time domain, the Pasupathy's MSK signal $s'(t)$ is given by equation that shows the OQPSK nature of MSK:

$$s'(t) = a_I(t)\cos\left(\frac{\pi t}{2\tau}\right)\cos(2\pi f_c t) + a_Q(t)\sin\left(\frac{\pi t}{2\tau}\right)\sin(2\pi f_c t) \tag{1}$$

The $a_I(t)$ and $a_Q(t)$ are two message streams, each of two possible values $\{-1, 1\}$. They are generated as a result of demultiplexing the bit stream $a_k(t)$. $a_I(t)$ and $a_Q(t)$ have the half bit-rate of $a_k(t)$ (Figure 13 (b)).

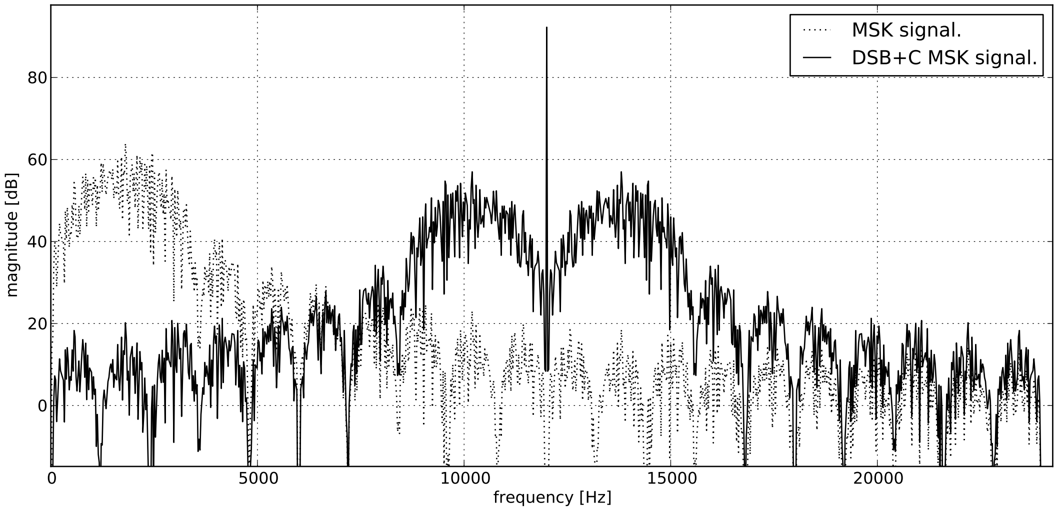
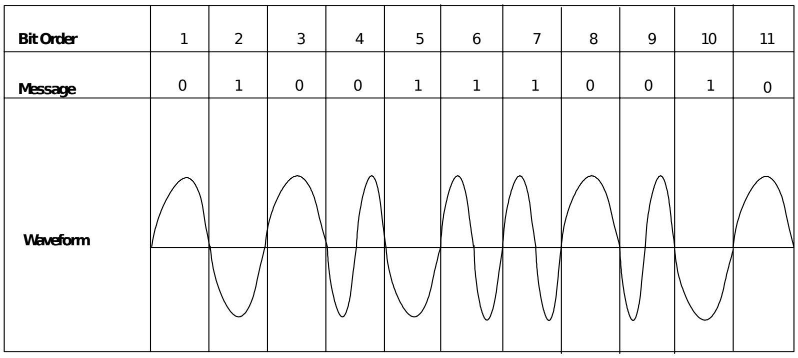


Figure **11:** The spectrum of result of simulation of MSK on AM: 12kHz sinusoidal carrier modulated using DSB+C AM, with amplitude sensitivity 0.9. The modulating signal is a random data Minimum-Shift Keyed on 1800Hz sine. Note the spike at the carrier frequency and two symmetrical side bands, which are an exact replicas of modulating signal. If 130MHz carrier was used, the spectrum would look similar — it would be only translated in frequency, but much more computation power would be needed to finish the simulation.



Bit Order	1	2	3	4	5	6	7	8	9	10	11
Message	0	1	0	0	1	1	1	0	0	1	0

Figure **12:** Waveform of the ACARS MSK. Image was taken from [ARNIC, 2008]

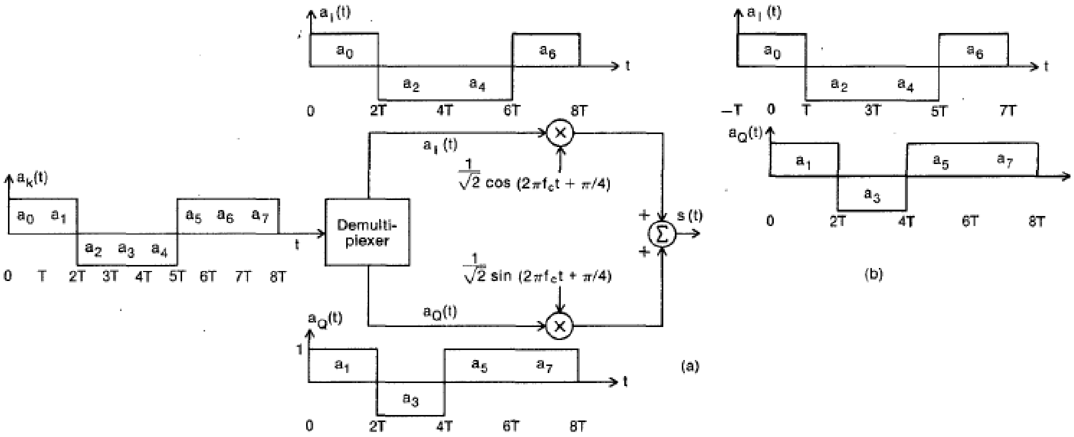


Figure **13:** (a) QPSK modulator. (b) Staggering of data streams in OPQSK.

The a_I and a_Q are interleaved NRZ coded bits of the message. For each ACARS transmission, the modulator starts with $a_I(0) = 1$, then the first bit is put to a_Q and the second to a_I and the third goes again to a_Q and so on.

The $\cos\left(\frac{\pi t}{2\tau}\right)$ and $\sin\left(\frac{\pi t}{2\tau}\right)$ terms are there because the MSK uses sinusoidal pulse signaling.

The $\cos(2\pi f_c t)$ and $\sin(2\pi f_c t)$ is the I-channel carrier and Q-channel carrier (in the sense of PSK, in FSK, the carriers have different frequencies).

Equation 1 can be rewritten ([Pasupathy, 1979] to the form of Equation 2, which better shows the way of demodulation of MSK as the FSK, because it clearly shows the frequency shifting:

$$s'(t) = \cos\left(2\pi f_c t + b_k(t)\frac{\pi t}{2\tau} + \phi_k\right) \tag{2}$$

The b_k is $+1$ when a_I and a_Q have opposite signs and b_k is -1 when a_I and a_Q have the same sign. ϕ_k is 0 or π corresponding to $a_I = 1$ or -1.[Pasupathy, 1979]. So with these substitutions we get:

$$s'(t) = \cos\left(2\pi f_c t - a_I(t)a_Q(t)\frac{\pi t}{2\tau} + \pi\frac{1 - a_I(t)}{2}\right) \tag{3}$$

Carrier frequency f_c must be chosen so that $f_c = \frac{1}{4\tau}n$, where $n \in \mathbb{Z}$.

4.1.2 The zero-crossings transitions MSK

ACARS uses a modified version of MSK the ”classic” MSK described by Proakis or Pasupathy — in ACARS, the change in the input bit stream results in the higher frequency symbol, and the same bits in the stream results in lower frequency symbols. This is satisfied if we say that the carrier frequency is $f_c = -1800$Hz. Please note the "$-$" sign. In fact, this is not a modification to the "classic" MSK, because it is only the result of tricky choose of $n = -3$.

The ACARS MSK has the bit transitions at the zero crossings, while the Pasupathy's MSK changes the bits when the signal is in it's lowest/highest point. Assuming $t = 0$ and $a_I(0) = a_Q(0) = 1$, the Pasupathy MSK signal is $s'(0) = 1$, while the ACARS MSK goes trough zero with same conditions. Next difference is that the ACARS MSK starts ($t = 0$) the transmission with positive slope (derivative) — assuming the $a_I(0) = 1$, the signal value rises for some amount of time just after the time $t = 0$.

This differences between the "classic" MSK and the ACARS MSK can be fulfilled by modification of the generating equation of the "classic" MSK to respect the requirements of ACARS MSK. If $\pi/2$ is added to the phase of the generating cosine:

$$\begin{aligned} s(t) &= \cos\left(2\pi f_c t - a_I(t)a_Q(t)\frac{\pi t}{2\tau} + \phi_k(t) + \pi/2\right) \qquad (4) \\ &= -\sin\left(2\pi f_c t - a_I(t)a_Q(t)\frac{\pi t}{2\tau} + \phi_k(t)\right) \qquad (5) \end{aligned}$$

Above was specified that

$$\phi_k(t) = \begin{cases} 0 & \text{for } a_I(t) = 1 \\ \pi & \text{for } a_I(t) = -1 \end{cases},$$

so we can write:

$$s(t) = -a_I(t)\sin\left(2\pi\left(f_c - \frac{a_I(t)a_Q(t)}{4\tau}\right)t\right) \qquad (6)$$

which is the equation of the time domain ACARS MSK modulated signal.

Choosing the $f_c = 0$ for the last equation, we see that if we do not count the symbol transition points, the instantaneous frequency offset from the carrier of $s(t)$ is $\pm\frac{1}{4\tau}$. In the transition points, the phase of $s(t)$ is not differentiable, and the instantaneous frequency is indefinite, because the frequency is a derivative of the phase. Despite that, the function is continuous in that points, because the symbol transitions happens in zeros of *sin*, no matter that the f_h or f_l was sent.

4.1.3 Constellation of MSK demodulated as FSK.

In FSK, the symbols are expressed as an energy of two orthogonal frequencies (not phases!). This orthogonality allows as to express the symbols as a Complex numbers, where the Imaginary part is amount of energy of one frequency and the Real part is the amount of energy of second frequency in the symbol time interval. The constellation for the FSK would be: $\{f_h, f_l\}$, or when expressed as the Complex number: $\{+1, +\mathbf{i}\}$.

In addition to the information from frequency of received waveform pulse, the MSK allows the receiver to get some information about the sent bit also from the phase of

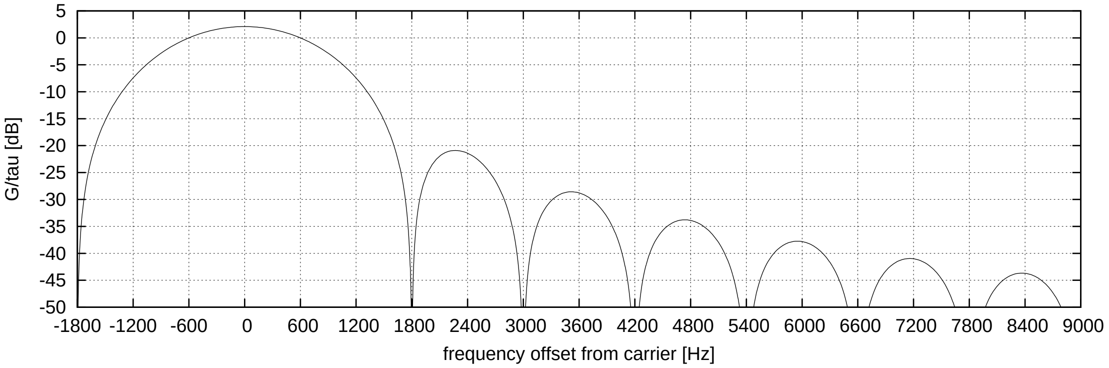


Figure **14:** Power spectral density of MSK around carrier f_c. The curve was plotted using $G(f)$ and $\tau = 1/2400$.

the currently received waveform pulse. The constellation of MSK is $\{+f_h, -f_h, +f_l, -f_l\}$, which can be also expressed in Complex numbers as $\{+1, -1, +\mathbf{i}, -\mathbf{i}\}$.

The symbols expressed as a Complex number will be needed further to construct a demodulating finite state machine. It also allows us to show the symbols on the Constellation diagram.

4.1.4 MSK spectrum.

[Pasupathy, 1979] specifies the MSK by formula, which was used to compute the spectrum on Figure 14:

$$\frac{G(f)}{\tau} = \frac{16}{\pi^2}\left(\frac{\cos 2\pi f\tau}{1 - 16f^2\tau^2}\right)^2$$

Although the signaling is done using two instantaneous frequencies, there are no spikes in the average spectrum of MSK. It is so because the data bits tends to be uncorrelated, random, so the instantaneous frequencies f_h, $-f_h$, f_l and $-f_l$ (the minus sign means the phase offset of π, an antiphased symbol pulse), tends to be equally distributed over the time in the $s(t)$. Each bit encoded as f_h cancels one bit encoded as $-f_h$ and same happens to f_l and $-f_l$.

Figure 15 shows the power spectral density (PSD) of ACARS MSK modulated signal. The second, dotted line is the PSD of so-called Sunde's FSK, which can be generated from the MSK by squaring the $s(t)$. This trick reveals the spikes in the average spectrum that were previously cancelled-out.

The MSK signal could be demodulated as Sunde's FSK, but the squaring amplifies the noise[de Buda, 1972], so the performance would not be optimal.

Remark 4.1 There is some mismatch between the Pasupathy's MSK and the MSK used in ACARS. The difference between the main lobe and the first side-lobe of the Pasupathy's MSK is lower, than in the estimated ACARS MSK. Also, the Pasupathy's MSK side-lobes rolled off $-20 - (-40) = 20$dB between 2400 and 7200Hz from the carrier, while

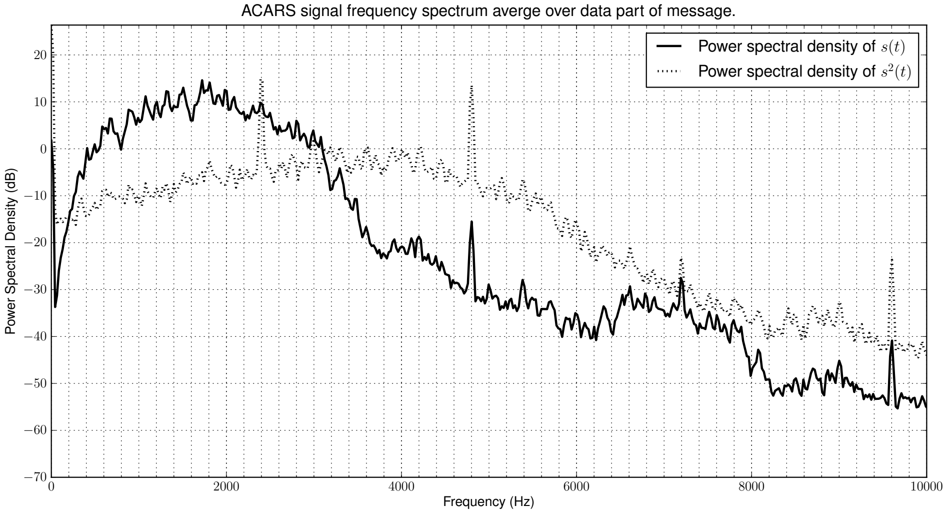


Figure **15:** Estimated Power spectral density of ACARS signal, computed over the payload part and square of the payload part of messages.

the estimated ACARS MSK rolled off about $-20-(-50) = 30$dB between frequencies $1800+2400 = 4200$Hz and $7200+1800 = 9000$Hz (the carrier is 1800Hz). I'm not sure why there is this difference. Possible reason is that the received signal was filtered. I guess that it also might come from the different position of symbols transition points. Anyway, designing the receiver, we don't have to bother with that too much. There is not much of energy in the side-lobes and it is common that they are intentionally cut off to not allow the noise to enter the demodulator.

4.2 Message format.

The ASCII charset (7 bit) is used to encode the message characters. Each character is protected with one parity bit, transmitted as it's 8^{th} bit (see subsection 4.3 for more detail information).

Every message is preceded with minimally 35ms signal of 2400Hz (pre-key). This signal can be used for locking the PLL for the higher frequency. Except of the pre-key, whole following transmissions is byte-oriented.

A `+` and `*` characters follows right after the pre-key. Except of first two bits of the `+` character and the last bit of the `*` character, they are keyed as continuous lower frequency with duration of 13 bits, so this part of message can be used for locking the PLL for the lower frequency.

Right after that, two characters `<SYN>` are transmitted. These are for character (byte start and end) determination.

After the `<SYN>` characters, a `<SOH>` character and the message payload is transmitted.

Pre-key	11111111111111111
`+`	1|0101011
`*`	0|0101010
`<SYN>`	0|0010110
`<SYN>`	0|0010110
`<SOH>`	0|0000001

Table **1:** Endoding of the ACARS message starting symbols.

There is a `<ETX>` or `<ETB>` character followed by 16 bits of CRC followed by a `<DEL>` character.

The starting part of the messages will be encoded (including the parity bit) as shown on Table 1.

The transmitted bit stream is shown on the Figure 16. We can compare it with real messages received, shown on Figure 17.

```
=== <Pre-key> ==|======= <+> =======|======== <*> ======|===== <Syn> =======|===== <Syn> ======|====== <SOH> =====|=========
   _____________|_____              |                ___|___   _       _____|__   _       _____|_    _____________|_____
…  111111111111 |  1 1\0 1 0 1 0 1  |  0 1 0 1 0 1 0/0  |  0\1/1\0 1 0/0 0  | 0\1/1\0 1 0/0 0  | \10/0 0 0 0 0 0  | V x x  …
                |      -------------|---------------    |    -   -----      |   -   -----      | --               |  ----
                |                   |                   |                   |                  |                  |
============================================================================================================================
```

Figure **16:** ACARS starting sequence. Characters, bit stream and instantaneous frequency (upper line is the f_h, lower line is f_l).

4.3 Error protection and bit transmission order.

Every transmitted 7-bit ASCII character (excluding Pre-key and CRC) is augmented with one odd parity bit as most-significant bit in byte. Bits are transmitted in such manner, that least-significant bit (LSB) is transmitted first, and most-significant bit (MSB) is transmitted last. This means that the parity bit is transmitted a last.

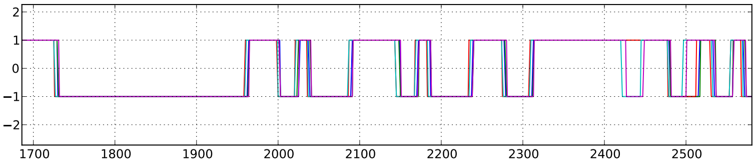


Figure **17:** Signum of instantaneous frequency offset from f_c of starting sequence of several messages. Value 1 represents 2400Hz, and value -1 represents 1200Hz symbol being transmitted.
Also, because I tried to align them on the 2000th sample, the increasing timing errors of can be observed at the right side of the plot.
The payload part of the message starts near the 2400th sample, so the transmitted symbols starts to differ from that point.

```
def update_CRC_byte( byte, crc ):
    crc ^= byte
    for i in range(8):
        if crc & 0x0001:
            crc >>= 1
            crc ^= 0x8408
        else:
            crc >>= 1
    return crc
```

Figure **18:** Method computing a CRC.

Except that the parity bits serves for the one-bit error detection, their presence also prevents transmitting of too long sequences of same bits, which would lead to the long sequences of same symbols being transmitted, which could cause the receiver to get out of sync. Because one odd parity bit is inserted for every 7 message bits, the longest sequence of same consecutive bits that may be transmitted is limited to 14 bits. An example of such sequences is the sequence characters `<SOH>` and `<NUL>` which produces sequence of 14 same bits as shown below (the emphaised bits are the parity bits):

Character	`<SOH>`								`<NUL>`							
Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Bits	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Messages are also protected with 16-bit CRC. CRC is computed from the message starting with `<SOH>` (not included), lasting with `<ETX>` or `<ETB>` (included). After `<ETX>` or `<ETB>` character, the 16 bits of the CRC register is transmitted without any parity protection. CRC is computed using CCITT polynomial $x^{16} + x^{15} + x^5 + x^1$, which is in binary (1) 0001 0000 0010 0001 = 0x1021 using MSB first, or 1000 0100 0000 1000 (1) = 0x8408 using LSB first notation. For every message, the CRC register bits are all preset to zero. CRC is computed from the message bits in same order as they are transmitted (LSB first).

4.3.1 Implementation of CRC algorithm.

The algorithm used to compute the CRC is fairy easy and well known. It performs the polynomial modulo-2 division of data by given polynomial. Only a bitwise operations and branching are needed for that.

The method shown on Figure 18 is called for every byte that might be protected with CRC. It first performs the XOR of whole CRC shift register with whole input byte, then if the shift register ends with 1 on the right, it is shifted to the right and then XORed with given polynomial, otherwise it is only shifted. When there is no more bytes to be checked, the CRC is the remainder after the polynomial division. We are shifting to right, so data comes from left into the CRC register.

5 Observed signal distortions.

I was given several wav files with recorded ACARS messages for the purpose of analysis and also for testing my demodulator. Two Citizen Band Radio transceivers with ability to demodulate AM were used to record them: Yaesu VR–5000 and Yaesu Pro–160. There were some distortions observable in some records:

- Nonconstant group delay.
- Nonlinear distortion.

The distortions are often harmful for the performance of the demodulator, thus it is desirable to avoid the distortion or try to correct it. The analysis of the distortions follows below.

5.1 Non-constant group delay.

Although the ARNIC specification [ARNIC, 2008] requires the ACARS modulators to ensure a symbols transitions to be exactly in the zero crossings, the received signals didn't have this property. This is probably caused by the channel delay that varied with a frequency (non-constant group delay), or the transmitters on the planes didn't work as specified by [ARNIC, 2008], which I see improbable.

The distortion is shown on Figure 19, where the line "received" is the received signal. There are also a sinusoids of frequencies f_h and f_l. Phase of these sinusoids have been shifted by a hand to match the "received" line as best as possible.

The symbol transition points can be found using the statement:

> The point at which the 2400Hz symbol changes to a 1200Hz symbol (and the reverse) can be found by noting that no discontinuities are produced by this distortion. Therefore, the 2400Hz symbol and the 1200Hz symbol amplitudes must be nearly equal at the point of symbol change. [Breitwisch, 1986]

The effect of non-constant group delay may be better observable on model shown on Figure 20.

Please note that the green and red lines (2400Hz and 1200Hz) doesn't cross each other in zeros. They don't even cross each other in ±1 values as the "classic" (Pasupathy) MSK signal does.

The effect of non-constant group delay can be removed by an adaptive equalizer. No of my experiments with GNU-Radio equalizers was successful — the counts of demodulated messages were lower than without the equalizer. I suppose it was caused by that the messages are quite short, so no equalizer was able to adapt quickly enough. Please note that this doesn't mean that more sophisticated equalizing cannot help.

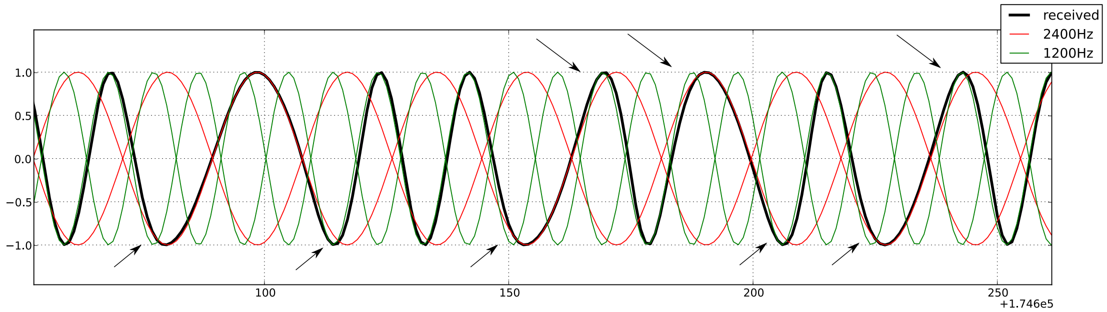


Figure **19:** Non-constant group delay distortion effect on the MSK. The possible symbol transition points are marked with arrows. Red and green lines are sinusoids with labeled frequency, phase-shifted by hand to fit the black (received signal) line as much as possible.

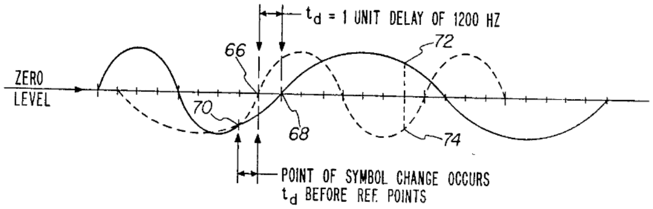

t_d = 1 UNIT DELAY OF 1200 HZ
66
72
ZERO LEVEL
70
68
74
POINT OF SYMBOL CHANGE OCCURS t_d BEFORE REF. POINTS

FIG. 4

Figure **20:** The effect of non-constant group delay on the MSK, described by [Breitwisch, 1986].

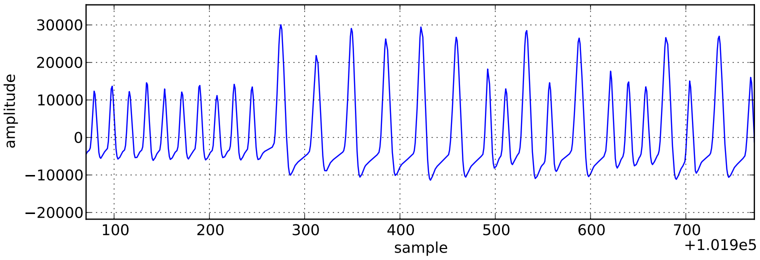


Figure **21:** Distorted start of the message.

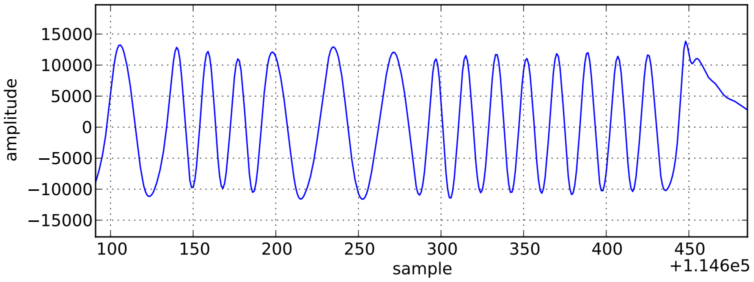


Figure **22:** Clean tail of the message.

5.2 Nonlinear distortion.

There was also a significant nonlinear distortion effects observed in the preamble of some messages. The effect was observable only on the message beginnings, where the envelope was still changing. All distorted messages were received using VR–5000 receiver, but not all messages received using VR–5000 suffered from that. There was no such distorted message received using Pro–160 receiver.

The Squelch function, that attenuates the noise when no signal is being transmitted, was turned off while recording, because it reacted too slow to let the message pass trough at all.

Example of the distorted message received using VR–5000 is shown on Figure 21. Clearly, the signal is periodic, but it is not sinusoidal. On the Figure 22 the clean end of the same message is shown. The signal there can be said to be much more sinusoidal thus not much harmonic distortion could occur at the messages end. The spectrum of the pre-key (Figure 24) shows significant amount of energy on frequencies of multiples of 2400Hz which is the fundamental. Similar effect occurs on the spectrum of the part of message containing 1200Hz signal, but the higher harmonics pikes are at multiples of 1200Hz (Figure 23).

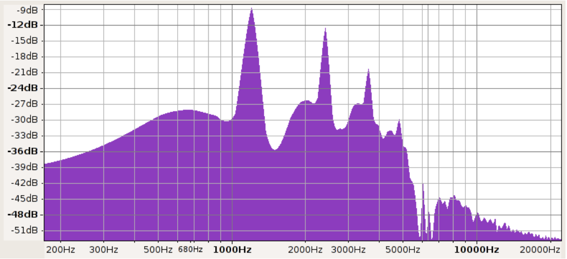


Figure **23:** Harmonic distortion of 1200Hz signal, logarithmic frequency scale.

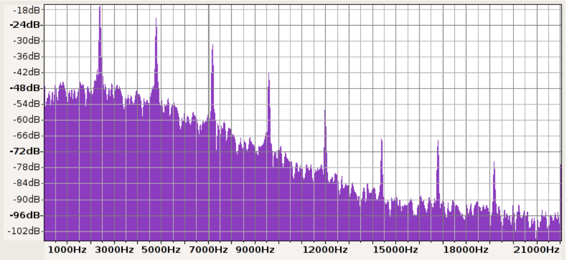


Figure **24:** Harmonic distortion of 2400Hz signal, linear frequency scale.

Remark 5.1 I have no message where the signal of frequency 1200Hz is long enough to show spectrum with sharp pikes. I also can't just take the distorted part of the message and show the spectrum of it, because the lines would disappear (see subsubsection 4.1.4).

This distortion is a serious problem for the demodulator. As it was said, it is present only on the beginning of some messages, but for the single-pass demodulator the beginnings are important to properly detect the presence of message and to synchronize. While the distortion of f_h symbol can be easily removed by a input filter sharp enough to not allow any harmonic (4800Hz, 7200Hz, ...) of f_h to pass, the signal of f_l symbol has the first harmonic of frequency f_h, which we cannot remove because by doing so we would loose the true f_h symbols.

Causes of this distortion remains uncertain, but I had some ideas where it can come from:

- Problem can be on the transmitter. This is in contradiction with the fact that no such distorted message was observed using Pro-160 receiver.
- The PLL of VR–5000, determining the AM carrier phase, may not be locking as fast as the PLL of Pro–160 does. Therefore there can be some phase difference between the real and the recovered carrier which causes the Q channel interfere with I channel in the receiver.
- Automatic gain control in the receiver is not fast enough to prevent a saturation and thus a compression (clipping) of signal. This causes the higher harmonic frequencies

to appear in the signal after the amplifier. However every harmonic that appeared as result of compression has same phase as the fundamental frequency, which is in contrast with that, what can be seen on the picture 21. Receivers can have multiple stages where the signal is amplified and there can be filters between them. These filters can cause not-constant group delay so they will delay some frequencies more, some less.

Remark 5.2 Receivers can contain equalizer to linearize receiver's phase response, thus the receiver can be said to have constant group delay, but because the higher harmonic appears on some intermediate stage in the receiver, it is not affected by all receivers amplifiers thus the equalizer may not compensate that.

6 Considered demodulator concepts.

6.1 Noncoherent detector

I experimented with noncoherent demodulator of MSK, which was based on a Hilbert transforming filter producing an Analytic (Complex) signal and function $\arg(\cdot)$ that gives an angle of a Complex number. A derivative of the phase (angle) is an instantaneous frequency, so differentiating the output of $\arg(\cdot)$ produces a signal that can be quantized and sampled in order to get the data. This method is very simple, because it doesn't need a carrier synchronizer, but the count of successfully demodulated messages was a half of my coherent demodulator. I refused this approach.

6.2 Correlating demodulator or matched filter demodulator.

Haykin (page 395 of [Haykin, 2001]) shows the schema (Figure 25) of correlating MSK coherent demodulator. There, the received signal is split into two branches and multiplied with frequencies signals $\phi_1(t)$ and $\phi_2(t)$ coming from some synchronizer. Then, filtering is done in both branches to remove harmonics produced by multiplying. Then the decision device slices the bits to ±1 which are then muxed into single bit stream.

Sometimes a matched-filter approach is shown in the literature — the multipliers and filters can be combined into one block by multiplying each sample of each filter impulse response with appropriate phase of sinusoid. This gives a system that produces identical result if sampled at correct time instants. [Haykin, 2001] This means the need of synchronizer remains.

The GNU Radio doesn't have any frequency divider, which is needed to make synchronizer shown in [de Buda, 1972] or [de Buda and Jagger, 1983]. I decided to build a demodulator using Costas loops, getting inspired by the patent [Breitwisch, 1986].

6.3 Two Costas loops.

Figure 26 shows the schema of Breitwisch's demodulator presented in the patent [Breitwisch, 1986]. It is slightly different approach than the OPQSK approach presented by de Buda, Haykin and Pasupathy. My demodulator performs quite the same actions, so it may be worthy to describe it.

In the Breitwisch's demodulator, the MSK is not demodulated as phase-keyed signal, but as the frequency-keyed signal. The input signal is split into two pairs of branches, each pair of branches is responsible for detection of presence and polarity of single frequency, either f_h or f_l

The pair of branches is multiplied (using the XOR gates) with the 0° and 90° signals from VCO. The frequency and phase of the VCO 0° output is maintained with a µP to match the estimated phase and frequency of symbol pulses of the received signal — the upper VCO is matching the $+f_h$ and $-fh$ symbols.

The I and Q counters in the pair acts like an integrators (low pass filters). The VCO phase error is detected by the Q counter and then minimized by actions on the VCO

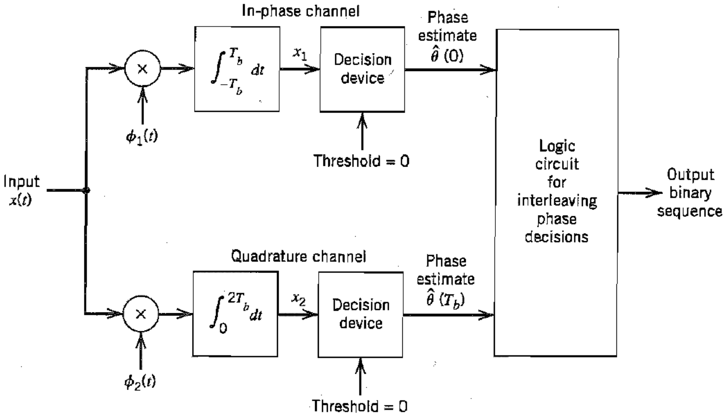


Figure **25:** Coherent MSK demodulator by Haykin[Haykin, 2001]

performed by the logic of µP. The bigger the error, the bigger is the absolute value in counter.

The other pair of branches does the similar job, so there is no need to describe them. Only difference is in the frequency that the branches are detecting. A pair of branches is basically a one Costas loop subsection 3.6.

The µP also maintains the clock and reset signal of the counters. The counters are reset in expected bit transitions and clocked often enough to meet phasing error limits. When phase of sampling frequency is not correct, the Q counters will both be positive or negative valued at the sampling time instant and this sign and value can be used to tune the frequency used to read and reset the counters. So the third feedback loop is introduced.

This design is advantageous in the fact that the µP is able to maintain the sampling of f_h and f_l symbols independently, thus nonconstant group delay effects can be eliminated. Unfortunately, I didn't find a way how this could be implemented with GNU-Radio without need of making a new signal-processing block.

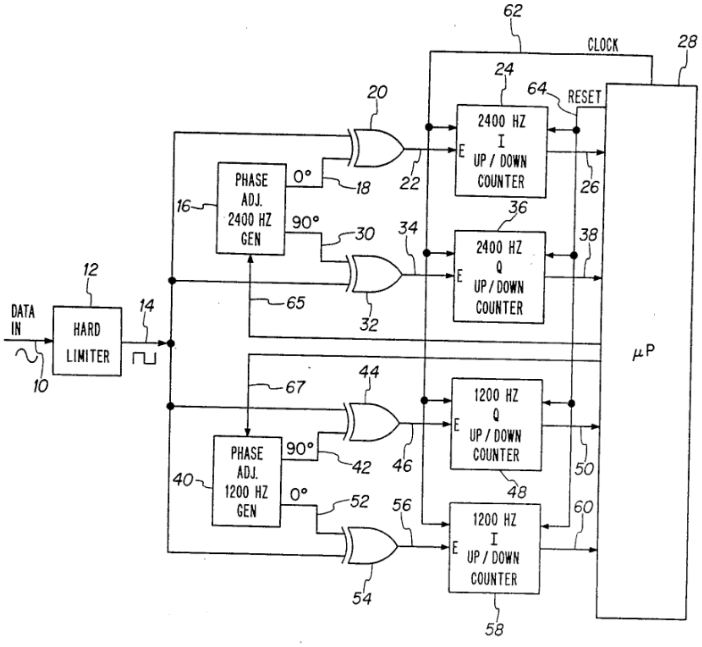


Figure **26:** Costas loops MSK demodulator[Breitwisch, 1986].

7 Demodulator

The approach of two Costas loops seemed to be applicable with GNU Radio, so I decided to follow this way. The schema of my demodulator is shown on Figure 27. The basics of it's operation is on Figure 28.

In short, my demodulator is basically two costas loops — one locks-on and detects the presence of f_h, the other one is for f_l. The demodulated symbols are then given as a input to a Viterbi algorithm performing a maximal likelihood estimation of the message. As it was said before, this demodulator was inspired by [Breitwisch, 1986], but changes had to be made to make it possible to be build using GNU Radio.

7.1 Input filter.

The input to the demodulator can be affected by a noise. Some part of the noise spectra will be out of the main lobe of the ACARS MSK spectrum. That would lower the demodulator performance, because it degrades the signal waveform. Looking at the plot of MSK power spectral density (Figure 14, Figure 15) we can see small side-lobes on frequencies more that 1800 Hz away from the carrier, so there may be more of noise energy than signal energy, therefore it seems wise to filter out these frequencies. [4] A linear filter is the tool to do that.

7.1.1 A FIR and linearity.

I used a FIR filter, because the linear phase response and thus also constant group delay of FIR can be achieved quite easy and the constant group delay means that the waveform will not be smeared by the filter.

FIR filters are not recursive. The non-recursive filters typically needs much more taps and thus they need much more of computation power than a recursive filters to perform similar job, but the linearity of phase response of recursive filters is not so trivial to achieve as it is with FIR filters.[Rader, 2006] If the computational power was a limiting factor, the input filter should be one of the first parts to optimize.

7.1.2 A FIR filter and the Hilbert transform.

Blocks following the input filter in the demodulator signal path will need an Analytic signal to work with. Analytic signal can be produced using a Hilbert transformer, which can be approximated by FIR filter. So we may first bandpass the input, then make a Hilbert transform of the result and make the Analytic signal. Or we might do all of it at once, because all operation that had to be done were linear. Both FIR were LTIS, so they can be combined.

The Analytic signal is a signal with no negative frequencies. Thus if we make a filter that has a passing bandwidth only at the positive frequencies, then we created a device

---

[4] Note that the difference of -40dB from some reference is the 100 times lower magnitude than the reference have.

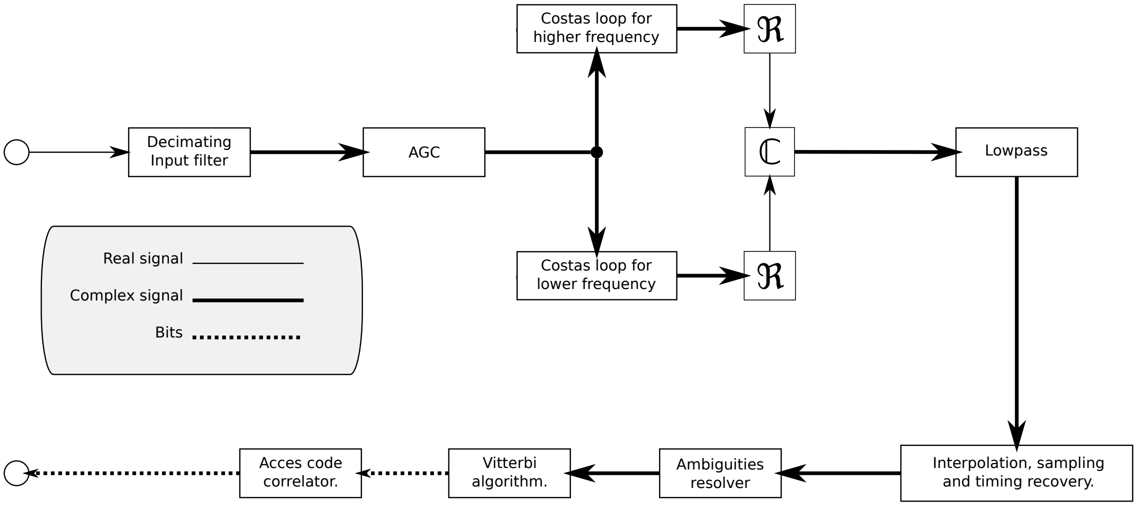


Figure **27:** Coherent MSK demodulator based on Costas loops.

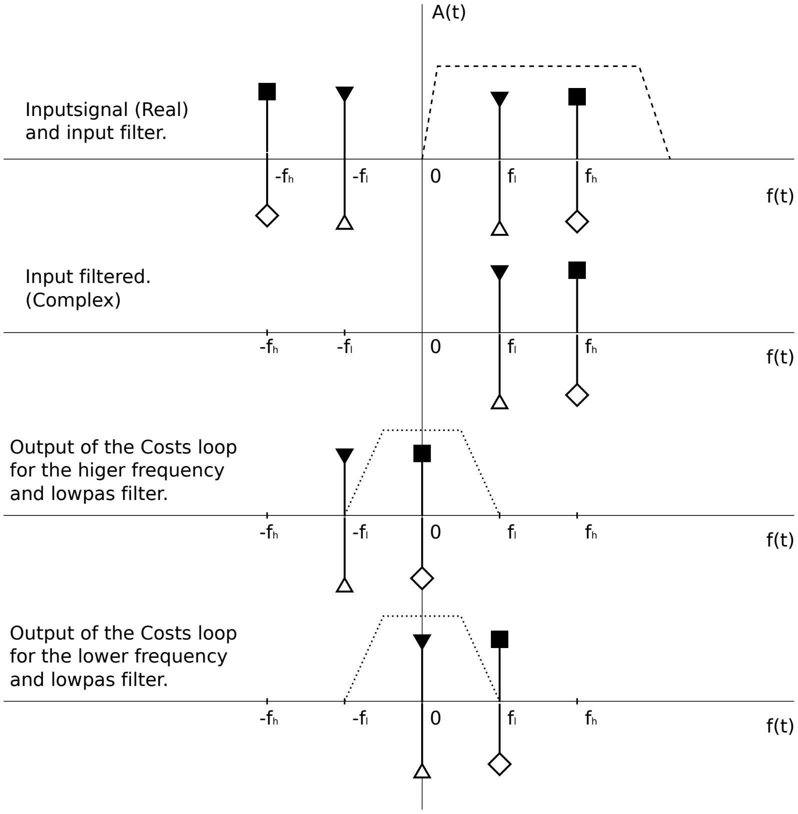


Figure **28:** The two Costas loops approach to demodulate FSK. Please note that this figure doesn't show an average spectrum, but the instantaneous frequencies, which better explains what is happening in the demodulator. In average spectrum, the MSK doesn't have any spikes.

producing an Analytic signal from the input Real signal. It is not possible to produce a Real-valued Impulse response filter with a passing bandwidth only in one side of the spectra, but if we allow the Impulse response to be Complex-valued, we can do that. This means that the output of such filter will be Complex-valued signal, because Convolution of a Real valued signal with the Complex valued signal is a Complex valued signal.

We will not loose any information by throwing the negative frequencies away, because the discrete Fourier transform of the Real signal is symmetrical, so we can reconstruct the thrown out part whenever we like. [5]

7.1.3 Designing the input filter.

For computing the taps of the input filter, the `gr_firdes.complex_band_pass_2` method is used. It can design a filter with the bandpass only on one side of the spectrum (Figure 29) so the result will be a band-limited Analytic signal. The filter on the figure was computed using parameters:

sampling rate = 9600
cutoff low = 300
cutoff high = 3300
transition width = 200
attenuation = 60

When the sampling rate was 9600Hz, the designed filter had 131 taps. When the sampling rate was 44100Hz, the filter had 601 taps.

7.1.4 Saving the computation power.

After removing the unnecessary frequencies, we can decimate the input to save computation power.

The function block used for the filtering also can decimate the input. Decimation factor is chosen to satisfy the Nyquist criteria with slight oversampling. The input filter has only one-sided spectrum, so the Nyquist rate is equal the bandwidth of the signaling. So the decimation factor (the ratio between the input rate and output rate) $d = \left\lfloor \frac{f_s}{o f_{hc}} \right\rfloor$, where f_{hc} is the higher cutoff of input filter and o is the oversampling factor. I got the best results with $o = 1.8$.

The filter implementation used is `gr_fir_filter_fcc` block. This block accepts stream of floats as an input and produces Complex numbers as an output. It must be preset with the Complex taps values (the impulse response of the filter), that we have computed above.

7.2 AGC.

AGC stands for automatic gain control. AGC in general is used to amplify or attenuate the signal to have predefined average amplitude.

---

[5] In reality we loose some information, because we can only approximate the Hilbert transformer. Nevertheless, the loose can be made very small.

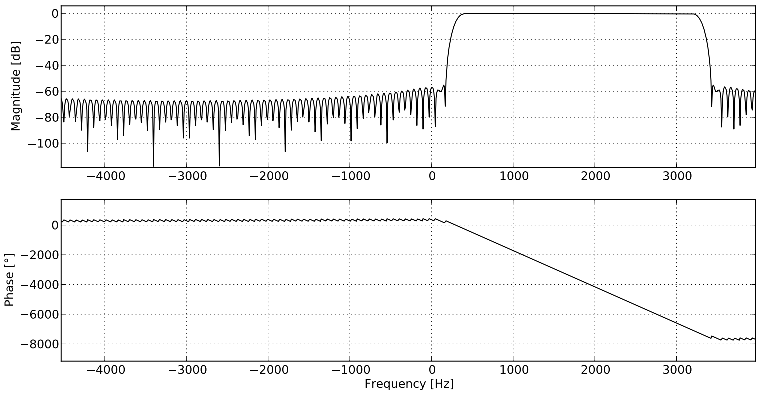


Figure **29:** Input filter frequency and phase response.

The envelope of MSK is constant. No information is in the amplitude of the signal so one might wonder why we need the AGC. It is because some blocks used in demodulator are designed to work with certain signal amplitude. For example the Costas loop, when using error function shown on Figure 9, will make greater correction steps when the input signal has greater amplitude. But we cannot be sure that the envelope of signal coming to the system does have constant envelope. We know that modulation produces signal with constant envelope and we don't want the system to be heavily influenced by the signal envelope, so we would like to equalize the envelope averaged over some time.

One way to do that is to use `gr.cma_equalizer_cc`, which is a constant-modulus equalizer — a FIR filter with taps updated by a value computed from the error feedback, which is the difference between modulus of output sample from the desired modulus. The taps are updated in the direction to minimize the error. The amount by which a tap will get changed is proportional to the value that the tap contributed to the output sample — the Least Mean Squares algorithm. Part of the GNU Radio code that updates the tap is:

```
tap += d_mu*d_error*abs(in)
```

`tap` is the tap being tuned

`d_mu` is a parameter changing the speed of reaction on the error

`d_error` is the difference between the modulus of filter output and desired modulus

`abs(in)` is the amount the tap has contributed to the output of the filter.

The best performance I got was with one tap and `d_mu` set to about 0.02.

More taps would be needed to compensate for some multipath signal propagation effects, but I guess that the ACARS messages are so short that the filter is not able to converge fast enough to gain some profit from more taps. I tried to experiment with more taps, but I didn't get better results.

7.3 Costas loops.

After AGC, the signal is ready for detection of presence of instantaneous frequency f_h or f_l. We need Costas loops (CL) to do that — Phase-lock loop won't work, because it cannot lock on signal, where one frequency with zero and opposite phase is present equally in average. If the PLL would fall into locked state, the short interval of opposite frequency would kick it out of the locked state.

The Costas loop for demodulating a BPSK is exactly what we need to detect a presence of frequency with phase ϕ or $\phi + \pi$. GNU Radio does contain such a block. It's name is `gr_costas_loop_cc`.

The output from the CL is Complex signal with envelope $A(t)$. If the frequency (f) and phase of signal coming to CL matches it's reference (f_0), the output of CL will be $A(t)+0\mathbf{i}$. If the input signal is in antiphase, the output of CL will be $-A(t)+0\mathbf{i}$. Otherwise a Complex sinusoid with frequency equal to the $f - f_o$ is present. [6]

7.4 Lowpass filter.

Because we want to detect a presence of the f_o, we can simply take the Real part of output of the Costas loop and lowpass it so we get $A(t)$. The moving average over τ time is the best filter to do that, because the $b_k(t)$ is either 1, or -1, so sudden frequency changes happens at bit transitions. The filter will also filter-out some of the noise inside of the ACARS MSK spectrum, that couldn't be removed by the input filter.

We could filter each Real part of CLs output separately, but the GNU Radio supports a Complex moving average, so I combined the two Real parts of Costas loops output to create a Complex signal, which is ± 1 when the f_h is present in-phased or antiphased or $\pm\mathbf{i}$, when f_l is present in/anti-phased.

7.5 Interpolation, sampling clock recovery and sampling.

The output from the moving average filter are Complex samples of rate $\frac{f_s}{d}$, the f_s is the sampling rate of signal to the decimation filter, d is the decimation factor chosen for the Input filter and the input rate. The signal coming from the filter is shown on constellation diagram Figure 32.

No bit-synchronization was done yet, so samples are not likely to be sampled at the constellation points of MSK detected with CL — $\{-1, 1, \mathbf{i}, -\mathbf{i}\}$, even that the signal passes through them. We need to compute the value of the signal between the samples we have (Figure 30).

Remark 7.1 Note that the Figure 30 may be misleading, because the optimum sampling time instants for my demodulator are the time instants where the signal from the lowpass filter has local extreme in Real part and zero at Imaginary part and vice versa — where it passes through $\{0, \frac{\pi}{2}, \pi, \frac{3}{4}\pi\}$ angles.

---

[6] The former expressions were just a special cases of the Complex sinusoid — a sinusoid of zero frequency.

We could have oversampled the signal on input to the demodulator, but that would lead to undesirable wasting of the processing power. [7] We also can interpolate between the samples going from the Costas loops. Interpolation is done by inserting a zeros between the samples of the signal and then low-passing it. It is better, because the CL's would be fed with lower sampling rate and only the clock recovery circuit would be fed with high rate. But this still wasteful, because lot of the interpolated samples would be thrown away in clock recovery, without any use, because only the sample near the optimum sampling time is needed. The `pfb_clock_sync_cc` is a cure for our problem.

The block was not well documented yet, but I have read the code and discussed with Tom Rondeau — the author of the block, and hopefully I can explain it's work.

The `pfb_clock_sync_cc` uses a polyphase filter bank approach to get near-optimum sample. It uses two sets of filters computed from the prototype filter which would be used for normal interpolating. One set are the interpolating filters, the second set is composed from derivatives of the first set. For each output sample, one of these filters is used for convolving the signal in convolution window. Window is moved between computing each of the output samples. The count of samples that the window is moved over and the filter used for computation of output sample, is chosen using standard, second order feedback loop. The error driving the loop is computed using the derivative of the interpolating filter — from the second filter set. Figure 31

In [Gaeddert et al., 2007], there is nice picture that may show the problem better than words, so I present it on Figure 30. The description of that picture follows:

> The matched filter output and the relationship between available sample points, optimum sample points, and interpolants.
>
> In this example the sample rate T_s is approximately twice the symbol rate, however the position of optimum timing slides to the right of the available sample points as time progresses indicating that the actual sampling frequency is slightly greater than 2 samples/symbol. Note that the bank consists of $M = 4$ filters. While none of the samples lie directly on the optimum sampling point, the resolution can be set sufficiently small by increasing M. [Gaeddert et al., 2007]

According to Tom Rondau, the modulation pulse shape filter should be used as an interpolation filter, which is some sort of sinc(t) function. Note, that the sinc function have maximum at symbol sampling time and it's derivative has zero at the same point.

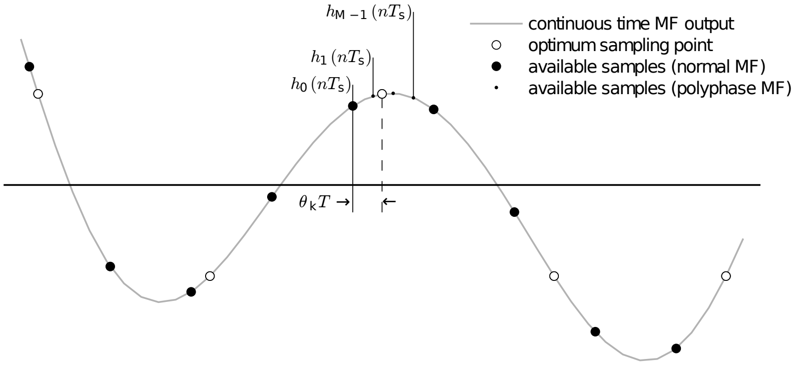
Let's denote i^{th} filter in N interpolating filters in bank with h_i. Each h_i is computed by taking (sampling) every N^{th} tap of prototype filter. The first filter in bank is the prototype filter sampled starting with the $N-1^{\text{th}}$ sample, the N^{th} filter is created using sampling with the first sample, so each filter in bank is rotated by $\frac{2\pi}{N}$.

The constructor of the block is:

```
gr_pfb_clock_sync_ccf::gr_pfb_clock_sync_ccf (double sps, float gain,
```

---

[7] An interpolation factor 128:1 is not an unrealistic demand. With a original sample rate of 9600k [samples/s the interpolated signal would have sample rate of 1.2M samples/s.



$h_{M-1}(nT_s)$
$h_1(nT_s)$
$h_0(nT_s)$
$\theta_k T$
continuous time MF output
optimum sampling point
available samples (normal MF)
available samples (polyphase MF)

Figure **30:** PFB interpolation: The samples doesn't have to be available at optimum sampling point, but using polyphase filter bank, the sample near the optimum can be very accurately interpolated. Image was taken from [Gaeddert et al., 2007].

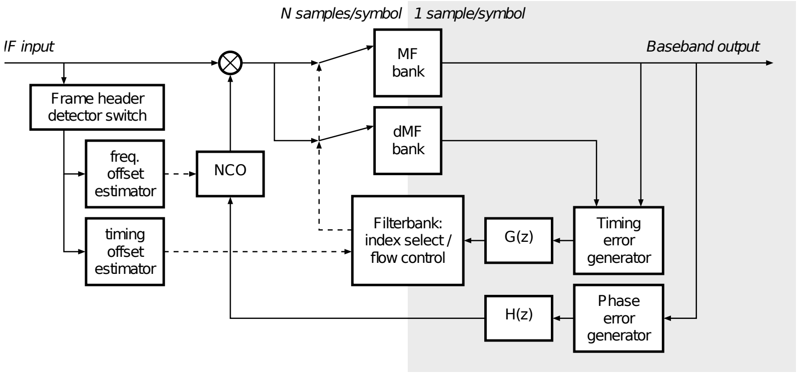


Figure **31:** The diagram of the polyphase filter bank clock synchronization. [Gaeddert et al., 2007]

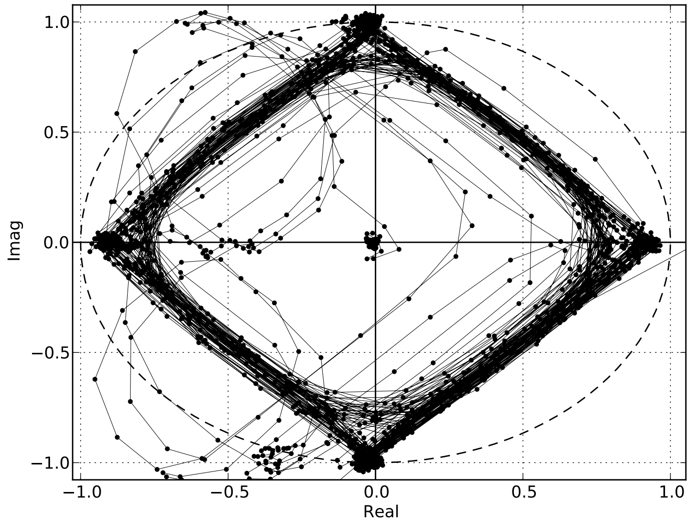


Figure **32:** Signal constellation with traces, taken after the moving averge (lowpass) filter.

```
const std::vector<float> &taps,
unsigned int filter_size,
float init_phase,
float max_rate_deviation)
```

`sps`	The expected samples per symbol ratio ($\frac{f_s}{2400 \cdot d}$)
`gain`	The first order gain (the phase error gain), second order gain can be set with `set_beta` method. Values around $1^{-}3$ are OK.
`taps`	The taps of the prototype of interpolation filter.
`filter_size`	The count of filters in the filterbank — N. Note that the name may be misleading.
`init_phase`	The phase of clock for the first symbol. Not used in my demodulator.
`max_rate_deviation`	Left at default value.

is each

Figure 32 shows the signal in from the moving average filter before the interpolation and sampling, the Figure 33 shows the signal constellation after interpolation and sampling.

7.6 Ambiguity resolving and message presence detection.

The polarity of the symbols from CL is ambiguous, because it the CL can lock either correctly phased or in antiphase. This ambiguity is removed by examining the known starting part of each message in Ambiguity remover block.

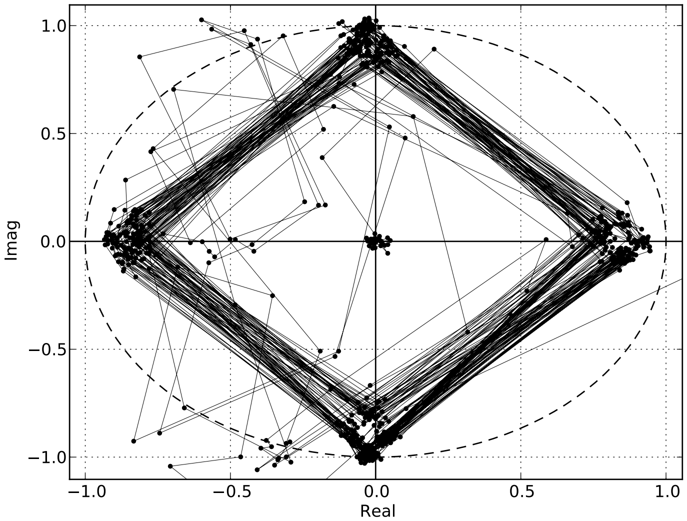


Figure **33:** Signal constellation with traces, taken after polyphase filter bank clock recovery.

7.6.1 How it might work.

We want the ambiguity remover to fix the sign of the Real part and/or Imaginary part of the symbols. This flip should take place soon enough — before we start receiving the data. So we need to find a rule usable to distinguish the data from the preamble.

The payload part of the message is protected with odd parity bits, thus no more then 14 same bits can be sent in consecutive sequence (subsection 4.3). Clearly, we can gain much from this. Because the same bits are encoded either as +1 or as −1 symbols, the absolute value of moving average over the Real part of more than 14 symbols of the correctly received data and parity bits can never be equal to 1 and must be less. This is not the case of message preamble, where the pre-key is not protected with parity bits.

7.6.2 Cross-correlation and it's relation to convolution and moving average.

Before I'll start describing how the ambiguity is resolved, I need to mention that moving average is the special case of cross-correlation, which is a linear operator $\star$ defined as

$$(f \star g)[m] \overset{\text{def}}{=} \sum_{n=-\infty}^{+\infty} f^*[n]g[n+m].$$

The f^* means Complex conjugation of f.

We can imagine the cross-correlation as that we Complex-conjugate a pattern f and then we begin to slide it from left to right, while incrementing m and keeping g on it's place. The m is the offset of the pattern f from the original position of f. Every slide we compute the sum of products of each value of slided f with g value and we write the result to $(f \star g)[m]$. The Complex conjugation ensures that when the Imaginary part of the pattern matches the Imaginary part of the signal g, we get positive addition to the sum $(-\mathbf{i} \cdot \mathbf{i} = +1)$.

If we define

$$\sqcap_l(m) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } |m| < \frac{l}{2} \\ \frac{1}{2} & \text{if } |m| = \frac{l}{2} \\ 0 & \text{if } |m| > \frac{l}{2} \end{cases},$$

then $\frac{\sqcap_l \star g}{l}$ is a moving average over l samples of g.

I may also mention that correlation $(\star)$ is very similar to convolution $(*)$ — when computing a convolution, one must first time-reverse the pattern and also no Complex conjugation is taking place:

$$(f \star g) \equiv (f^*[-m] * g[m]),$$

thus correlation with some finite pattern can be done using FIR filter, because FIR filter performs the convolution. As a convolution kernel we use the reversed, conjugated pattern.

7.6.3 Seeking for the match.

We can correlate received symbols with some pattern and according to the correlation coefficient we can flip the received symbols before they will be passed for processing in the trellis decoder. When designing a correlation pattern, we have to keep in mind that the signal may contain both frequencies at once although MSK doesn't allow that. The harmonic distortion may take effect. We shouldn't decide to flip or not-flip the symbols dimension only according to the correlation in same dimension, but the correlation strength of the second dimension should be taken in account as well. The picture Figure 34 shows the absolute value of Real and Imaginary channel after Costas loops and filtering. We can clearly see that in the leading part of the message, which is critical for synchronization, the harmonic distortion takes effect so when the Imaginary channel magnitude is high, significant amount of energy lays also in the Real channel. The distortion vanishes near the right side of the plot.

It is possible to detect the presence of a pre-key and some part of the symbol sequence generated by `+` and `*` in a symbol stream correlating the symbol stream with patterns:

$$p[n] = \begin{cases} 1 & \forall n \in \langle 0, l_h) \\ \mathbf{i} & \forall n \in \langle l_h, l_h + l_l) \\ 0 & \text{otherwise} \end{cases},$$

how to choose the l_h and l_l will be explained later.

If we take a look on the cross-correlation of p^* with p (denoted as $p^* \star p$) and the, so called, auto-correlation of p — the cross-correlation of p with itself (denoted as $p \star p$) shown on Figure 35, we may notice that the function that created the strongest and sharpest result was the auto-correlation. The result has a peak exactly where the pattern matches best. This can be used to determine the position of the best match.

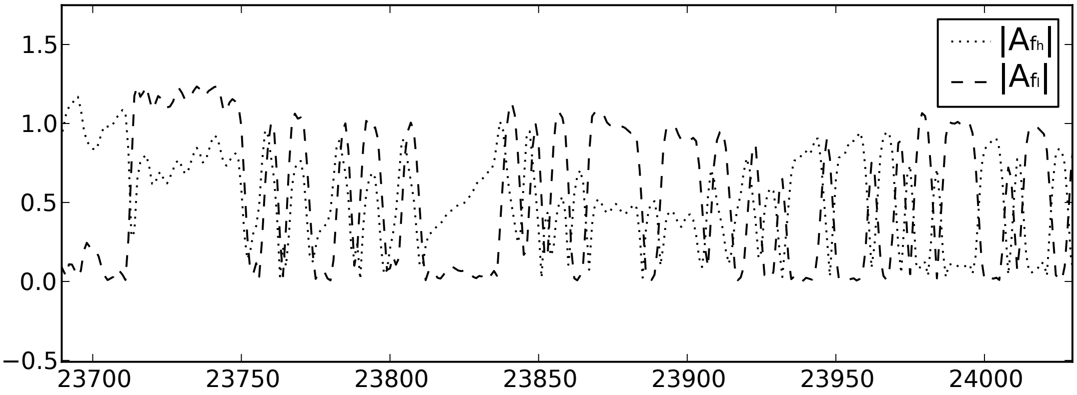


Figure **34:** The harmonic of f_l channel could be mistakenly interpreted as f_h energy: The $|A_{f_h}|$ line shows the absolute value of amplitude of signal coming from the CL for f_h frequency symbols, the $|A_{f_l}|$ line show the output of CL for the f_l frequency symbols.

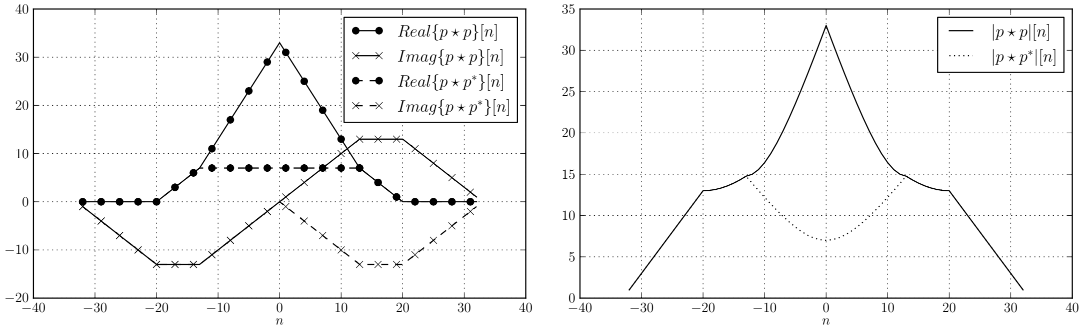


Figure **35:** Pattern p correlating with itself and itself's conjugation.

From the Figure 35 we might see that we can be interested only in th the Real part of the cross-correlation, because at the points near $n = \pm 10$, the absolute value has a value 15 while the Real part is only below 10 and the peak of the both is same.

To compute only the Real part, we can use theorem:

Theorem 7.1

$$\mathbb{R}\{f \star g\} = (\mathbb{R}\{h\} * \mathbb{R}\{g\}) - (\mathbb{I}\{h\} * \mathbb{I}\{g\}),\ \textit{where}\ h[n] = f^*[-n]$$

Proof. We will use the relation of cross-correlation to convolution and the properties of convolution. The convolution is associative, distributive and commutative linear operator, so we can:

$$\begin{aligned}
\mathbb{R}\{f \star g\} &= \mathbb{R}\{h * g\} && (7)\\
&= \mathbb{R}\{(\mathbb{R}\{h\} + \mathbf{i}\mathbb{I}\{h\}) * (\mathbb{R}\{g\} + \mathbf{i}\mathbb{I}\{g\})\} && (8)\\
&= \mathbb{R}\{\mathbb{R}\{h\} * (\mathbb{R}\{g\} + \mathbf{i}\mathbb{I}\{g\}) + \mathbf{i}\mathbb{I}\{h\} * (\mathbb{R}\{g\} + \mathbf{i}\mathbb{I}\{g\})\} && (9)\\
&= \mathbb{R}\{(\mathbb{R}\{h\} * \mathbb{R}\{g\}) + (\mathbb{R}\{h\} * \mathbf{i}\mathbb{I}\{g\}) + (\mathbf{i}\mathbb{I}\{h\} * \mathbb{R}\{g\}) + (\mathbf{i}\mathbb{I}\{h\} * \mathbf{i}\mathbb{I}\{g\})\} && (10)\\
&= \mathbb{R}\{(\mathbb{R}\{h\} * \mathbb{R}\{g\}) + \mathbf{i}(\mathbb{R}\{h\} * \mathbb{I}\{g\}) + \mathbf{i}(\mathbb{I}\{h\} * \mathbb{R}\{g\}) - (\mathbb{I}\{h\} * \mathbb{I}\{g\})\} && (11)\\
&= (\mathbb{R}\{h\} * \mathbb{R}\{g\}) - (\mathbb{I}\{h\} * \mathbb{I}\{g\}) && (12)
\end{aligned}$$

■

So we see that to get the Real part of the correlation, we only need to convolve the Real part of pattern with Real part of incoming signal and Imaginary part of the pattern with Imaginary part of the incoming signal.

Then we can compute what we will get when the signal x is flipped wrong.

$$\begin{aligned}
h_R[n] &= \mathbb{R}\{p^*[-n]\} = \mathbb{R}\{p[-n]\}\\
h_I[n] &= \mathbb{I}\{p^*[-n]\} = -\mathbb{I}\{p[-n]\}\\
\mathbb{R}\{p \star x\} &= (h_R * \mathbb{R}\{x\}) - (h_I * \mathbb{I}\{x\})\\
\mathbb{R}\{p \star -x\} &= (h_R * \mathbb{R}\{-x\}) - (h_I * \mathbb{I}\{-x\})\\
&= -(h_R * \mathbb{R}\{x\}) + (h_I * \mathbb{I}\{x\})\\
&= -\big(+(h_R * \mathbb{R}\{x\}) - (h_I * \mathbb{I}\{x\})\big)\\
&= -\mathbb{R}\{p \star x\}\\
\mathbb{R}\{p \star x^*\} &= (h_R * \mathbb{R}\{x\}) - (h_I * -\mathbb{I}\{x\})\\
&= (h_R * \mathbb{R}\{x\}) + (h_I * \mathbb{I}\{x\})\\
\mathbb{R}\{p \star -x^*\} &= -\mathbb{R}\{p \star x^*\}\\
&= -(h_R * \mathbb{R}\{x\}) - (h_I * \mathbb{I}\{x\})
\end{aligned}$$

It is possible to check whether the x correlates with any of the patterns p, p^*, $-p$, $-p^*$ — to search for the position of the pattern in x, the sum absolute values of the of the

Real part of convolutions can be used. This will ensure that the parts are always added together and thus the peak value will appear no matter how the constellation is flipped:

$$\Lambda \overset{\text{def}}{=} \left|h_R * \mathbb{R}\{x\}\right| + \left|h_I * \mathbb{I}\{x\}\right|$$
$$\Lambda \geq \left|(h_R * \mathbb{R}\{x\}) - (h_I * \mathbb{I}\{x\})\right| = \left|\mathbb{R}\{p \star x\}\right|$$

If we let x be some realisation of the message symbol's sequence, then when $\Lambda[t] \approx \mathbb{R}\{(p \star p)[0]\} \gg 0$, we have found a match in time t. To determine what channel should be flipped to fix it we can compare

- $0 < h_R * \mathbb{R}\{x\}$ and $0 < -h_I * \mathbb{I}\{x\}$. Then we have found a match in time t, both Costas loops are correctly phased, because the match is with the pattern that is the pattern of correctly detected message symbols. So when this happens, nothing have to be done with incoming symbols.
- $0 > h_R * \mathbb{R}\{x\}$ and $0 > -h_I * \mathbb{I}\{x\}$, both Costas loops are phased wrong, so we need to multiply the incoming symbols sequence with -1 and also Complex conjugate the result.
- $0 < h_R * \mathbb{R}\{x\}$ and $0 > -h_I * \mathbb{I}\{x\}$, we must Complex conjugate the symbols, and
- $0 > h_R * \mathbb{R}\{x\}$ and $0 < -h_I * \mathbb{I}\{x\}$, we have to multiply the symbols with -1.

We would like only one of these cases happen at time when receiving the leading part of the message, and none of them must happen when we are receiving the data. We should define some threshold to compare the Λ with.

As can be seen from Figure 35, when $\mathbb{R}\{p \star p\}$ is maximum $\mathbb{R}\{p^* \star p\}$ is much less. A little more than $(p^* \star p)[0]$ should be such threshold to not get false positive matches. Bigger the threshold, the stronger correlation is required for match to be detected, which means a better fit is required.

The correlation is a linear operator, therefore it's output magnitude is proportional to the magnitude of it's inputs, so some sudden envelope changes, which the AGC couldn't filter out, would confuse the comparators deciding if the correlation is strong enough.

To not let the sudden changes in the amplitude of the input signal to confuse our ambiguity resolver, it will be good to decide the symbols before correlating them — decide which of the $+1, -1, +\mathbf{i}, -\mathbf{i}$ is on the input of the Ambiguity remover. A GNU Radio `constellation_decoder_cb` can do that for us.

7.6.4 Choosing the length of the correlation patterns (l_h and l_l).

To filter out as much noise as possible — the "noise" caused by data or the Real noise, which is present when no message is being transmitted, we must maximize the l_h. Optimally, it is equal to minimal length of pre-key minus the average time the Costas loop needs to lock. We want the correlation to be strong when correlating the pattern with pre-key and in same time we want the correlation not to be too strong when correlating the pattern with data. Too big l_h may cause imprecise detection of the prekey, when the message doesn't contain a sequence of f_h symbols longer than l_h. In that case the noise that was present in on the channel before the pre-key started would be in correlation window as well as the pre-key unnecessary lowering the performance.

According to [ARNIC, 2008], the maximal pre-key length is 85ms which is about 200 symbols. The Costas loop detecting the f_h should get locked very quickly: 5 symbols should be enough. Nevertheless the envelope of received signal while receiving the prekey may change a lot before it settles, so much lower part than 200 symbols should be used. By several experiments I found that the value l_h about 60-90 worked best with the records I have been given.

l_l should be set to 13, because this is the count of same symbols that follows the pre-key — the time when the f_l frequency will take the biggest part of the received signal energy.

Taking more of the bits after the `+` and `*` characters is possible but I wouldn't recommend to do that if the symbol sampling clock is not driven directly from phase-lock loops (as the synchronizer in the article[de Buda, 1972] allows). When generating the clock from symbol transitions, sampling the symbols may be imprecise in the beginning. Some symbol could be doubled or skipped which will cause seriously mismatching pattern — the `<SYN>` character symbols have sharp autocorrelation, because theirs bit values vary much, thus the performance of ambiguity detection would suffer much from sampling frequency errors. The leading part of message consists of a sequence of same symbols, so missing or skipped symbol is no worry.

7.6.5 Example match.

The plot on Figure Figure 36 shows the example of Real and Imaginary parts of correlations normalized by the energy of the pattern to make the peak maximum equal 1. $\frac{(p \star x)[n]}{50+13}$, and $\frac{(p^* \star x)[n]}{50+13}$, where:

$$x[n] = \begin{cases} 1, \text{ for } n \in \langle 0, 50) \\ \mathbf{i}, \text{ for } n \in \langle 50, 50+13) \\ \text{a random number from the set } \{1, +\mathbf{i}, -1, -\mathbf{i}\}, \text{ for } n \in \langle 50+13, \infty) \end{cases}$$

$$p[n] = \begin{cases} 1, \text{ for } n \in \langle 0, 20) \\ \mathbf{i}, \text{ for } n \in \langle 20, 20+13) \end{cases}$$

The l_h was set low for this example, so the data noise is not damped too much. Nevertheless, the peak can be clearly observed on sample 30, where the pattern p^* perfectly matches the signal x.

7.6.6 Implementation of the Ambiguity resolver.

Entire Ambiguity resolver is implemented using Python to wire couple of the GNU Radio blocks together creating a one larger block. A schema of the wiring is on Figure 37.

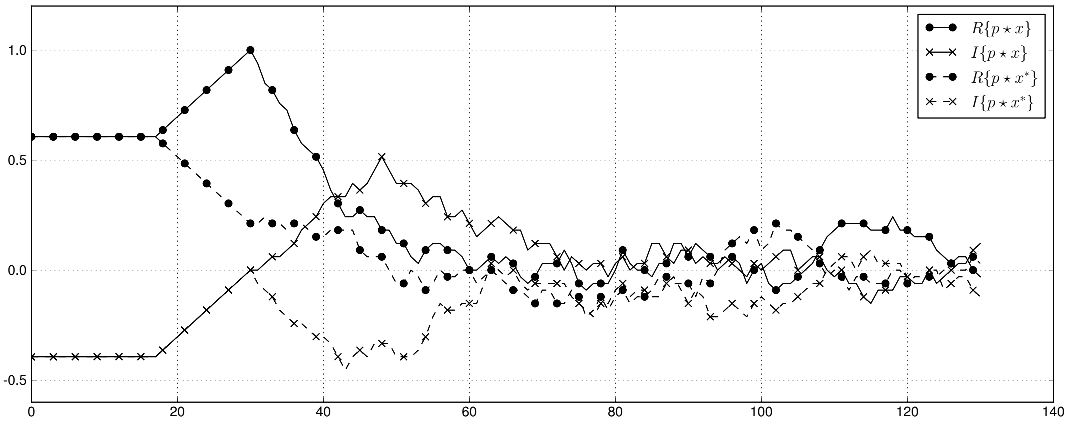


Figure **36:** Correlation (normalized to make output equal 1 when the best possible match occurs) of pattern p with signal x. It can be used to find the start of the message and to resolve phase ambiguities.

The idea is to split the signal to Real and Imaginary part and use the fact that we can use the sign of the correlations of pattern with the signal to flip the symbols as needed: $\forall x \in \mathbb{R} : x \cdot 1 = x = (-x) \cdot (-1)$.

Walking trough the diagram Figure 37 we find, that at first the input symbols are sliced to $+1, -1, +\mathbf{i}, -\mathbf{i}$, then the Real part is separated from the Imaginary part and both branches go to the recursive moving average filters. [8]

After the moving average, the Real part branch is delayed by l_l. That is to compensate the fact, that in the pattern that we want to match — p, the sequence of 1 lays on the lower indexes than the sequence of $\mathbf{i}$, so in the convolution kernel created by time-reversing and conjugating the pattern p, the sequence of 1 is time delayed after the sequence of $\mathbf{i}$ by l_l. The two moving averages filters computes the convolution with two ”partial” kernels, but without the time shift. Thus before the partial convolutions will get summed together, the shifting must be made. We are using the superposition and shift invariance properties of the convolution operator.

The two absolute value blocks, together with the summation, are computing the function Λ. Then the signal goes to the peak detector and a thresholding function block.

The peak detector outputs binary zeros until the peak is detected. On the sample which is the peak it produces a binary one. After the peak it again keeps giving zeros on the output. The peak detector could report a peak even if the correlation was weak so the binary AND is used to suppress the false correlation peaks. The second operator for the AND block is a signal that comes from a thresholding block giving zeros until the Λ is large enough.

---

[8] Moving average filters can use a recursive algorithm. Recursive method will be more efficient than the ”straightforward” non-recursive method. [Smith, 1998].

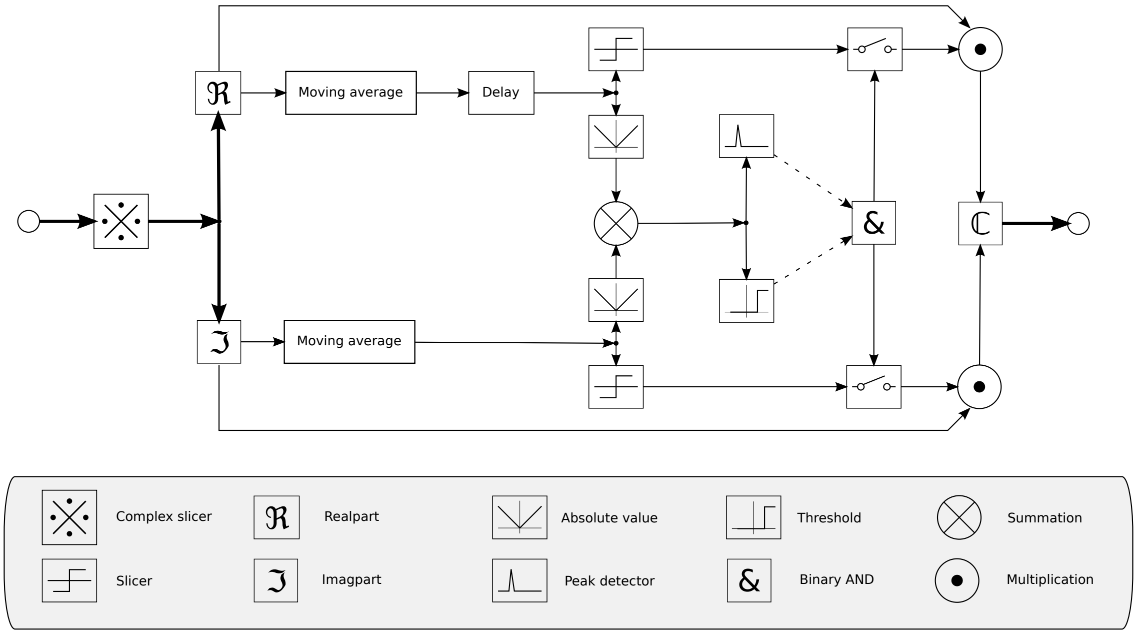


Figure **37:** Symbols polarity ambiguity resolver schema.

The two identical binary signals from the AND block drives the sample'n'hold blocks, which passes the limited signal coming from the moving averages as long as the binary ones arrives to it's gating control input. When a zero is on the gating control input, then the sample'n'hold holds the sampled value until the next binary one arrives.

The idea, how the flipping is done, is that the Real part branch moving averaged signal will be greater than zero, when no flip in the Real part of the incoming signal must be done. The same counts for the Imaginary part. The result of the sample'n'holds is used to flip the incoming signal.

7.7 Viterbi algorithm.

Now, when ambiguity of the symbols has been removed, we can map the symbols onto the stream of bits.

7.7.1 The meaning of the keying symbols.

In each symbol, there isn't only the information about current bit being sent, but also about the previous one. Suppose we denote the last bit of pre-key as odd$^{\mathrm{th}}$, then when receiving the symbol:

$+1$	we know that current bit is 1 and the previous one was 1 as well.
$+\mathbf{i}$	the current bit is negation of the previous bit, or it's value can be computed also from the current bit parity — if the current bit is even$^{\text{th}}$, it's value is 0 if it is odd$^{\text{th}}$, it's value is 1.
$-\mathbf{i}$	the current bit is negation of the previous bit, or it's value can be computed also from the current bit parity — if the current bit is even$^{\text{th}}$, it's value is 1 if it is odd$^{\text{th}}$, it's value is 0.
-1	means that the current bit is zero and the previous one was zero as well.

So for decoding one symbol it is good to know something about the history. We see that the modulation follows some grammar (in communications it is said that the modulation has a memory). Taking an advantage of knowledge of the grammar complicates the decoder, but such decoder will produce less mistakes than the simple constellation decoder would.

7.7.2 Easy way.

The easy way how to decide received symbols and output a message bits is to use simple constellation decoder. Constellation decoder slices the constellation plane into segments. Then, depending on which segment the received symbol lays in, the output bit is generated. This is easy, but it doesn't take any advantage of the knowledge of the grammar the modulation uses and thus it cannot be the optimal way to decide the symbols.

The problem of the constellation decoder is that, when the signal is bad due to low signal-to-noise ratio or other disturbances, the decoder may be given symbols which are near the decision line and some may even happen to get over the line. Then the constellation decoder will likely make errors.

7.7.2.1 Computing a distance Suppose we change the constellation decoder to not produce bits, but a distance of the received symbol from each of the constellation points. The distance of the received symbol from constellation point p may be a good measure of how much we can be sure that the transmitted symbol was the point p. We will use this later.

7.7.3 Recognizing the language of MSK.

In terms of Computation theory, the MSK modulator produces symbols that are from Regular language. We can construct a Finite state machine (FSM) that recognizes any sequence the MSK modulator can produce.

The FSM is a base for the Finite state transducer (FST) that can translate the sequence of input symbols into the bits. When it is talked about the FSM, the FST is often meant.

The problem of ordinary FST is that it decides what was the sent symbol, changes its state, and produces output depending only on its current input and current state. The future decisions are heavily dependent on the current decision because the currently wrong decision means that FST choose wrong transition, and probably ends up in different state than it should, and will probably produce wrong output. A possible way how to deal with this is to simulate any decisions it can make. So we will construct a decision tree.

Remark 7.2 If we explore all possible paths, we cannot miss the best path.

7.7.4 Decision tree.

Each node In each level of such tree represents the state the FST might be in. For every new received symbol x for every node s in the leaves layer of the tree we construct an edge e that models a possible transition $s \xrightarrow{p} s'$ of the FST from the state s to the state s' if p was on the input of the FST. We evaluate that edge e with a distance of the x from the constellation point (p). How to compute the distance we deduced in paragraph 7.7.2.1.

Then, at the end of the message, the decoder can choose that path from the root of the constructed tree that best fits the received signal. It can be done simply summing the costs of edges from each leave of the tree to the root, and choosing the cheapest one.

7.7.5 State explosion.

If we denote m to be the number of ways the FST can go for every symbol to be decoded and n the count of the symbols, we see that although the supposed algorithm is an optimal solution for demodulating a sequence of symbols (in terms of bit error ratio), it consumes $O(m^n)$ bits of memory, so it must consume also at least $O(m^n)$ operations of computation time, thus it is possible to do only for small m and n, but it doesn't mean that we cannot perform maximum-likelihood decision for long streams. To avoid the explosion, the computation tree must be pruned to not allow it to spread too much.

7.7.6 Pruning the tree.

So far we were creating a new node in the decision tree for each transition of FST. Let's suppose that in every state of the FST there are two transitions to another states in FST. In our tree it would be modelled as that there are at-least two edges going from each of the nodes in the tree. So every input symbol causes addition of one layer to the tree with two times more nodes than previous layer.

Then, because the number of states of the FST is finite and the nodes in the tree represents the states of the FST, after some count of input symbols, on some level of the tree, some of the nodes will represent the same FST state, and their subtrees will be also identical for every next symbol. These nodes can be joined together, so the unnecessary redundancy won't appear in the tree.

Because the FST has finite number of states, and we don't allow more than one node representing the FST state to appear in one level in the tree, the number of nodes in each level of the tree is limited to the count of FST states. We see that the number of nodes the tree will have after receiving n symbols is bounded to $O(|Q|n)$, where Q is the set of states and the state explosion won't happen.

This way we created a structure called Trellis — a special kind of directed acyclic graph. Now we want to choose the cheapest path in such graph.

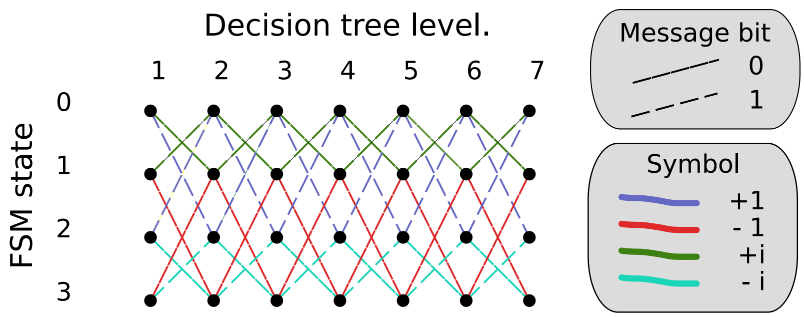


Figure **38:** Trellis diagram for the ACARS demodulator with Costas loops. In any time — any decision tree level, the ACARS modulator's FST can be in one of the four possible states. The Viterbi demodulator tries to estimate the path trough the trellis which the modulator's FST could go while encoding the message bits into the keying symbols using the sequence of observations — the received, noised and distorted symbols. If such path is known, the message bits can be easily derived from it.

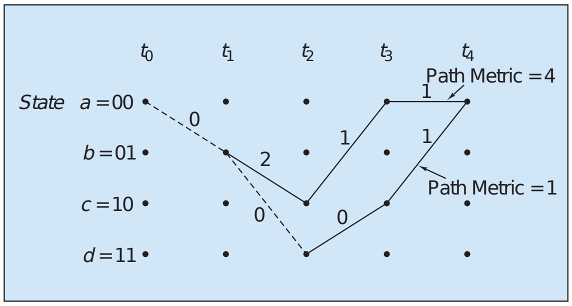


Figure **39:** The simplified view on the trellis showing the selection of the better path. The dots in columns represents the nodes in same level of the decision tree. The numbers near the edges are their costs. The bottom path (a, b, d, c, a) has a lower metric (cost), so it wins. Image taken from [Sklar, 2003]

Suppose we are constructing a cheapest path from node S_i to node S_j. There are two paths in the graph, one from node S_i through some sequence of nodes, continuing to the node S_j. The other path goes from node S_i through some other sequence of nodes and then to node S_j *forming a circle*. These two paths have different costs. Both of them cannot be a part of the cheapest path from S_i to S_j, no matter which nodes we select to continue the path with. Only the shorter one can. Therefore we can drop all the information about the more expensive path at every branch joining and continue constructing only with the cheaper path.

7.7.7 Putting all together.

We derived the way to not allow the state explosion to happen by pruning the decision tree by transforming it into the trellis. We derived that we won't loose any information by pruning the tree, and we derived that we are able to select the path in the trellis that is the maximally likely estimates the transmitted symbols. We created the maximally likelihood estimator of the sequence of symbols so it can decide what was transmitted keeping track of some part of the history. If that part of the history is long enough, our estimator will surely be close the optimal estimator. We derived the basics of Viterbi algorithm.

Remark 7.3 Nice source of information about the Viterbi algorithm is [Sklar, 2003].

7.7.8 The implementation

The GNU Radio has the Viterbi algorithm implemented. The block that may demodulator uses is `trellis.viterbi_combined_cb`. It is a distance measuring block and the Viterbi algorithm combined into one block. The constructor of this block takes several arguments:

`FSM`	The finite state transducer used to *modulate* the message.
`K`	The number of symbols in one block for which maximum-likelihood estimation is performed for. The Viterbi algorithm will keep track of $|S| \cdot$ `K` paths.
`S0`	The initial state of the modulator's FST for every block. In the ACARS decoder the state on the beginning of the block is unknown, because it is not ensured that the block starts on the beginning of the message, so I used `-1` for this parameter, indicating that any state of the FST is possible to start with.
`SK`	The final state of the modulating FST. Again, `-1` is used in my ACARS demodulator.
`D`	Dimensionality. How many `TABLE` elements describes single constellation point. 1 is used here, because I use Complex numbers as the constellation points and my demodulator symbols are two-dimensional, so one Complex number describes one point.
`TABLE`	The table of constellation points. I use `[ -1, -1j, +1j, 1 ]`.
`TYPE`	The type of metric for computing the distance of symbol from the constellation point. `trellis.TRELLIS_EUCLIDEAN` gave the best performance so I use it.

7.7.9 Viterbi finite state transducer design.

GNU Radio requires a model of FST that is hypothetically used for the modulator. I used the word "hypothetically", because there are more ways of creating the MSK signal; without using the FST.

When constructing a demodulator, the FST for translating the symbols back to bits is actually needed. It might be confusing that GNU Radio Viterbi decoder needs the modulating FST. GNU Radio will construct the "demodulating" FST itself from the given "modulating" FST.

The required FST is passed to the initializing method as its parameter `FSM`. A Mealy-type transducer is required — this means that it must compute the output based on it's current state and the current input.

$x(i-1)$	$\rho(i-1)$	$S(i)$	$x(i)$	$S(i+1)$	$y(i)$
1	odd	0	0	1	$+\mathbf{i}$
1	odd	0	1	2	$+1$
0	even	1	0	3	-1
0	even	1	1	0	$+\mathbf{i}$
1	even	2	0	3	$-\mathbf{i}$
1	even	2	1	0	$+1$
0	odd	3	0	1	-1
0	odd	3	1	2	$-\mathbf{i}$

$x(i-1)$	is the $(i-1)^{\text{th}}$ message bit.
$\rho(i-1)$	denotes the parity of the message bit — whether it is even$^{\text{th}}$ or odd$^{\text{th}}$.
$S(i)$	is the state of FST when i^{th} bit is being encoded/decoded. GNU Radio methods accept only numbers as state identifiers, so I present it here for completeness even that the $\big(x(i-1), \rho(i-1)\big)$ tuple is enough to represent the state.
$y(i)$	is the keying symbol for the i^{th} bit.

Table **2:** The FST transition table.

I designed the required modulating FST using the information from [ARNIC, 2008]. It's transition diagram is presented on Figure 40 and transition table on Table 2.

The output of the Viterbi algorithm block is a stream of bytes which is an unpacked form of the message — only the least significant bit of every output byte represents the message bit — 8 stream bytes are needed for one message byte.

7.8 Access code correlator.

The Viterbi algorithm block outputs a stream of bits — the demodulated message, but the messages starts and ends must be found prior to the displaying the message. I used the `correlate_access_code_bb` block to mark the starts of the messages.

The input to the Access code correlator is a stream of bytes, where the LSB of every byte represents one message bit. The output is the same stream of bytes, with small distinction — the second LSB (match flag) of each byte is also set/cleared, depending on whether the access code matches the input. The match flag is set in the byte following the last matching byte.

The Access code correlator shifts a window with predefined pattern over the input stream and counts how many bits of the windowed sequence differs from the pattern (the Hamming distance). The flag bit in the output stream is cleared to zero if the distance is above certain threshold, otherwise the flag bit is set. Figure 41.

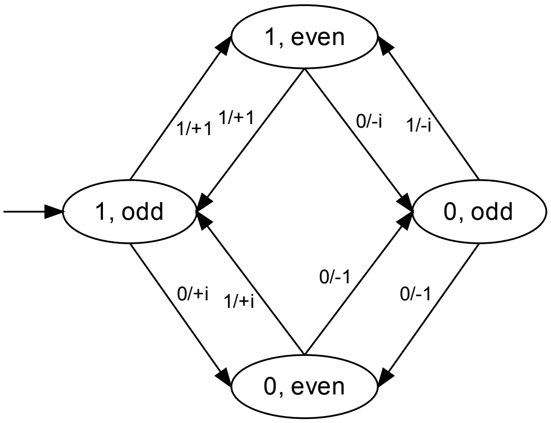


Figure **40:** Finite state transducer used to encode or decode ACARS messages. Labels on the edges represents $x(i)/y(i)$. The states are denoted by the $x(i-1)$, $\rho(i-1)$.

```
Character      |<SYN>  |<SOH>  |
Time      |0   |5   |10  |15  |20  |25
Input      xxxxx0110100010000000xxxxx
Pattern         011010001000000
Flag       00000000000000000000100000
```

`x` is any random bit. Note that the bits are shown in transmission order — LSB first, and also note that only 7 bits of the `<SOH>` are in the current correlation window position. The MSB of `<SOH>` will be matched later.

Figure **41:** Correlating the Access code example.

Acces code	Threshold	CRC OK messages	Parity error messages
`01101000 1000000`	1	171	2000
`00 1000000`	0	176	7750
`0 1000000`	0	176	17700
`1000000`	0	176	30800

Table **3:** Access code experiments.

7.8.1 Choosing the best access code.

I was experimenting with various access code correlator settings. The results are on Table 3

Most of the parity errors in my demodulator are caused by false-positive message start detections between the messages, when a noise is being received — we don't mind that. However the false-positive detection of message start, several bits before the correct start, can lead to the parity errors and loss of whole message. But this is not likely to happen because the pre-key is a stream of constant ones — a sequence very distinct to the one that `<SOH>` will be encoded to. Two `<SYN>` characters are also very distinct to the `<SOH>`, so I did not try to build a demodulator avoiding this phenomena.

I used `0 1000000` as the access code. It is the LSB bit of the `<SYN>` character and 7 MSB of the `<SOH>` character. A compromise had to be considered: The preamble of the messages tends to be distorted so the preamble bits tends to be demodulated wrong. After the start of the transmission, the messages are considerably cleaner bit after bit, so it is reasonable look for the match as late as possible. But too low count of the correlating bits will rise the risk of premature message start detection. The `<SOH>` is the last bit that is (by the information contained in [ARNIC, 2008]) the last character usable to perform the message start detection because it is contained in every ACARS message on same position. The experiment results shows that the 1 bit of `<SYN>` and 7 bits of `<SOH>` are a good compromise. (Figure 34.

There are only 7 bits of the `<SOH>` in the access code that I chose. GNU Radio will set the flag bit in next byte after the last matching one, so the flag bit will indicate the last transmitted bit of the `<SOH>` character. When the byte in the output stream of this block have the message bit set to 0 and the flag bit set to 1, it probably is the last transmitted bit (parity bit) of the `<SOH>` and thus the message starts with the next byte in the block output stream.

7.9 Passing the message bits to the Python code.

Not all of the work is done in the GNU Radio (GR). Some smaller part, which doesn't demand much of computation power, is left for processing in Python, but then we have to pass the data to the Python code somehow.

Passing the data to through a pipe looked as the best way to do that. The GNU Radio offers a `file_descriptor_sink`; that sink writes it's input to the file descriptor. A writing end of the pipe is passed to the `file_descriptor_sink` to write to. The Python code reads the data from file descriptor specifying the reading end of the pipe.

The Python code that does the rest of the processing is run in new thread and is started after the GR processing is started. The GR code runs in several threads. There is a scheduler in GR that organizes all of the work that lays on the GR. All the GR processing is started by running a `start` method of the block which describes the signal flow diagram (in Python). The `start` method doesn't block. It returns immediately.

Several threads are created by the GR itself. One thread is created by my code for the final processing of the messages in Python. The main thread only starts the GR and the thread for final processing and waits for the GR to finish, than it closes the writing end of the communication pipe. This will signal the EOF to reader of the pipe.

After closing the writing end of the pipe, the main thread waits for the code for final message processing — the reader — to finish. When the this code gets to the EOF (which was caused by closing the pipe) it finishes. The main thread ends after joining the final message processing thread. It will close the reading end of the pipe and exits.

7.10 The final message-processing.

Final message processing is done completely in Python. The data are read from the file descriptor. Each byte is examined and then dropped if not a part of a message. If the byte read from the descriptor is equal to 10_2 (see the Access code correlator in subsection 7.8), it means it contains the last bit of `<SOH>` character. This means the code for final message-processing should stop dropping the further bytes and start analyzing them more closely.

When starting receiving the message, CRC register is reset, control is passed to the loop that packs the bits to the bytes (it puts the MSB to the left), checks the parity of each byte, updates the CRC register and then sets the MSB of the packed byte to zero. This loop is repeated as long as the parity matches or the `<ETB>` or `<ETX>` character is found. These characters indicates the end of the payload of the message and the start of the 16 bits of the CRC followed by the `<DEL>` character.

If the 16 bits of CRC matches the CRC register, the message is passed to the formatting code that parses the content and prints it out. If the CRC doesn't match, the message is discarded.

7.10.1 Finding the message ends.

The reception of ETX or ETB character results the demodulator to receive the CRC and continue finding the next message start.

A mistakenly received ETX or ETB can result the receiver to remain in wrong state, receiving a garbage, but this situation is not likely to last long. Every character of the ACARS message is protected with a parity bit. The garbage character is half likely to be parity-correct. Only one bad parity character causes the message to be flagged with bad parity flag and it's reception is interrupted immediately, continuing by finding the start of next message.

7.11 Demodulator output.

My demodulator is capable to produce three formats of output: A human readable format, a comma separated values format, and raw format. The output format can be chosen with the `-f` command line argument: `-f hr` specifies the human readable, `-f csv` the CSV and `-f raw` the raw format. `-f hr` is the default.

Each output format contains a date and time of demodulation of each message. Unfortunately only some messages includes the time reference, so the time of demodulation is printed. The UTC timezone is used to report the times of the messages to prevent confusions caused by daylight saving times and leap years.

7.11.1 Human readable format.

Appendix A shows an example of my demodulator output. Several messages are duplicated in the listing, probably because the intended receiver didn't get the message, so it didn't sent the ACK, so it was repeated.

Each message is separated from others with a line of ”`=`” signs in the output, then follows the headline, showing the time.

Then comes the well structured message data from header and part of the message text transformed to human readable form.

`Label`	describes the type of message,
`Flight`	is the shortcut for the airline followed by the flight number,
`Address`	is a callsign assigned to the airliner.
`Mode`	When it is set to `VHF Category A`, the message is broadcast for all ground stations. `VHF Category B` means that the message is intended for only one Datalink service provider.
`Originator`	specifies which device produced the message.
`Direction`	indicates whether the message comes from the airliner or the ground,
`UDBI`	An Uplink/Downlink block identifier.
`TechAck`	will contain `NAK` if the message is not a response of some previously sent message. `NAK` can also really mean a negative acknowledgement — a response generated for example when the message was received with errors.
`MesSeqN`	A message sequence number.
`BlockSeqN`	A block sequence number.
`Suf`	Suffix. It can contain either `ETX` while this message is the last in the block sequence (End of text), or `ETB`, which stands for ”End of transmission block — the text of the message is spread over more blocks that are enumerated with BlockSeqN. Each message block sequence contains a block with `ETX` suffix indicating that it is the last one.

The messages also contains a Text part. It is also very often structured, machine generated and could be transformed into human readable form, but the format is often proprietary and varies message type to message type, airline to airline. Only the first 10 characters of each downlink message should specify the UDBI, MesSeqN, BlockSeqN, the airline and the flight number, so this information is extracted and displayed in corresponding fields. The rest of the Text is displayed in the Free text field. [ARNIC, 2008]

7.11.2 The comma separated values format.

CSB format is good for further machine processing of the messages. It contains collumns:

```
utc_year, utc_month, utc_day, utc_hour, utc_minute, utc_second,
label, t_label, ali, fn, address, mode, t_mode, originator,
t_originator, direction, ubi_dbi, msn, t_msn, bsc, t_bsc,
tech_ack, free_text, t_end, raw
```

Some columns have a prefixed variant (`t_`). Both of them contains the same information, but translated to human readable form. The last column (`raw`) is the whole message in raw format. It is there to let the another parsing code to parse the messages itself to get some missing information.

7.11.3 The raw format.

The raw format contains only the full date and time and raw message.

The date and time is in the UTC timezone and it is formatted as specified in ISO ISO8601. For example `2010-07-20 20:23:44.213460Z`. The `Z` on the end means ZULU, which is the indicator for UTC.

After the final `Z`, the `<Tab>` character is printed (`U+0009`), then follows the raw message, starting by the `<SOH>`, ending by the `<ETX>`, or `<ETB>`. The CRC and final `<DEL>` is not included.

8 Installation and use.

8.1 Python

My demodulator works with Python v2.6. I will not describe how to instal Python, because it should come with any Linux distribution.

8.2 GNU Radio

GNU Radio have to be correctly installed before the use of my demodulator. I decided to use the `pfb_clock_sync_cc` block, which is a new block, not contained in the last released version of GNU Radio, so a development version of GR have to be compiled and installed. The revision, that I have tested my program with, is specified by it's git SHA hash `278b6db3de99ae31d5f4f79dcd62708c4757d7fa`. [9]

To download, build, and install, the following sequence of commands should be executed:

```
git clone http://gnuradio.org/git/gnuradio.git && cd gnuradio
git checkout 278b6db3de99ae31d5f4f79dcd62708c4757d7fa
./bootstrap
./configure --prefix=/usr/local
make
su
make install
ldconfig
```

The GNU Radio compilation takes about a half hour on Intel Pentium M processor 1.73GHz, 2GB RAM machine. On my system, the `make` command finished with an error when generating a documentation for GNU Radio, but despite that, after `make install` the the GNU Radio worked.

After installation, depending on the configuration of the operation system, it might be needed to specify the search path for the python modules by setting the environment variable.

`export PYTHONPATH=$PYTHONPATH:/usr/local/lib/python2.6/site-packages/`. You may want also to put that line somewhere in your shell init script (`~/.bashrc`)

8.3 pyacars

pyacars itself doesn't need any special handling and can run "out of the box", however, to allow it to run from any directory, an environment variable must be set to let the `coherent.py` know where to find the dictionaries and FSM:

`export PYACARS_PATH=/opt/pyacars/`. You may want also to put that line somewhere in your shell init script (`~/.bashrc`)

---

[9] There is a repository snapshot on the attached CD.

8.4 Usage.

While trying to run the demodulator without parameters, it will ask for the source to read from:

```
pyacars % ./coherent.py
Usage: coherent.py { wav WAV_FILE | stdin SAMPLE_RATE |
rec SAMPLE_RATE [ DEVICE_NAME ] }

coherent.py: error: Please, specify the INPUT_TYPE.
```

If we want to record from the sound card with (unnecessarily high) sampling rate 44100Hz, we can use:

```
pyacars % ./coherent.py rec 44100 default
```

For reading fom wave file and for raw messages output:

```
pyacars % ./coherent.py wav ../sounds/test.mix.wav -f raw
```

For reading binary float numbers from stdin file:

```
pyacars % ./coherent.py stdin 44100 < sound.raw
```

`Ctrl + c` will interrupt the processing and print the summary info.

Remark 8.1 Example outputs are in attachment of this document.

8.5 SOX

SOX is not integrated in my demodulator, but it is very handy to perform some actions. GNU Radio has problem reading some Waveform Audio File Format (wav) files generated with some tools (audacity). The error message `RuntimeError: is not a valid wav file` appears. GNU Radio also cannot read compressed files like FLAC. These problems can be solved using SOX as preprocessing tool. For example, demodulating from FLAC file can be done using

```
sox ../soundfile.flac -t f32 -r 9600 - | ./coherent.py stdin 9600
```

The `-r 9600` specifies that the `sox` input will be filtered and resampled to frequency 9600Hz, `-t f32` ensures the 32-bit floating point samples and `-` (dash) makes SOX output using std. output, which is piped with the demodulator std. input. The demodulator only takes 32-bit floats when using std. input. Finally, the 9600 on the end is, of course, the sample rate of the input stream.

9 Comparision of the results.

I tried to demodulate several files with several demodulators:

- SkySweeper 3.13 Demo Licence
- ACARSD
- Airnav Acars Decoder 2.0

Neither the Airnav Acars decoder, nor the ACARSD can demodulate from a recorded file, so I used capturing the PCM to device of the sound card in Microsoft Windows and playing the sounds in Microsoft Media Player.

pyacars was tested under Linux where this functionality doesn't work for me. Anyway, I made the for of the pyacars little harder, I connected the headphones hole with the line-in hole with the metallic cable and I set the computer mixer to record from the line-in. Despite of that, it's result were much better than the results of other demodulators.

The demo version of SkySweeper cannot record from the sound card and is limited in size of file we want to demodulate from, so I shrunk the noisy space between messages to make the message density higher.

	SkySweeper 3.13	ACARSD	Airnav Acars Decoder 2.0	pyacars
test.mix.wav	69	73	57	107
test.pro160.wav	0	15	11	14
test.sndfile.wav	14	34	8	34

Table **4:** The numbers in this table are the counst of sucessfully demodulated messages with matchig CRC.

10 Conclusion

The demodulator I designed is capable of real time demodulation of the ACARS messages from the sound recorded using the computer sound card or from specified wav file, or from standard input. The demodulated messages are displayed on the standard output in human readable, comma separated values, or in raw format.

Although the demodulator performs well, there still is some space for some improvements. The parameter values of some blocks could be better designed. Also, it is not able to deal with the harmonic distortion present in some of the messages. Despite that, the short comparison with another ACARS demodulators showed, that this demodulator can perform much better then the others.

The demodulator doesn't consume much of the processing power when run in real time. Surprisingly, I would like to make it more computationally demanding — to correctly demodulate as many messages as possible. Now, the demodulator tries to demodulate each message only once. If the CRC of demodulated message doesn't match, demodulator discards it.

Jaroslav Henner

11 References

[ARNIC, 2008] ARNIC (2008). Strawman 3 of draft 2 of supplement 7 to ARNIC specification 618, air-ground character-oriented protocol.

[Boccuzzi, 2008] Boccuzzi, J. (2008). *Signal Processing for Wireless Communications.* McGraw-Hill.

[Breitwisch, 1986] Breitwisch, R. L. (1986). *Phase distortion adaptive detector of minimum shift keying data.* US Patent 4 569 061.

[de Buda, 1972] de Buda, R. (1972). Coherent demodulation of frequency-shift keying with low deviation ratio. *IEEE Transactions on Communications*, pages 429–435.

[de Buda and Jagger, 1983] de Buda, R. and Jagger, C. E. (1983). *Self-Synchronization circuit for a FFSK or MSK demodulator.* US Patent 4 384 357.

[Gaeddert et al., 2007] Gaeddert, J., Volos, H. I., Cormier, D., and Reed, J. H. (2007). Multi-rate synchronization of digital receivers in software-defined radios. *SDR Forum Technical Conference.*

[Haykin, 2001] Haykin, S. (2001). *Communication systems.* Wiley, fourth edition.

[Haykin and Veen, 1999] Haykin, S. and Veen, B. V. (1999). *Signals and systems.* Wiley.

[Mattos, 2009] Mattos, A. (2009). SITA AIRCOM service (VHF & satellite) — safety in the air 2009.

[Oishi, 2002] Oishi, R. (2002). FANS is seven years old. *The global link*, 23.

[Pasupathy, 1979] Pasupathy, S. (1979). Minimum shift keying: A spectrally efficient modulation. *IEEE Communications Magazine*, pages 14–22.

[Proakis and Manolakis, 2006] Proakis, J. G. and Manolakis, D. K. (2006). *Digital Signal Processing (4th Edition).* Prentice Hall, 4 edition.

[Rader, 2006] Rader, C. M. (2006). The rise and fall of recursive digital filters. *IEEE Signal Processing Magazine*, pages 46–49.

[Sklar, 2003] Sklar, B. (2003). How i learned to love the trellis. *IEEE Signal Processing Magazine*, pages 87–102.

[Smith, 1998] Smith, S. W. (1998). *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing. [Online; accessed 29-April-2010].

[Various authors, 2002] Various authors (2002). Various articles. *The global link*, 23.

[Wikipedia, 2010] Wikipedia (2010). Aircraft communications addressing and reporting system. [Online; accessed 24-April-2010].

A Dem. example outputs.

```
pyacars % sox ../sounds/ACARS_vr5000_ATT_RF128_recout.flac \
-b32 -e float -t raw -r9600 - trim 0 1:04 | \
./coherent.py stdin 9600 -f hr
Using decimation factor 1.
>>> gr_fir_fcc: using SSE
>>> gr_fir_ccf: using SSE
>>> gr_fir_fff: using SSE

================================
ACARS Message decoded at [ Čt 22. červenec 2010, 19:54:38  UTC ]:
Label:        1L
Flight:       MS 0732
Address:      SU-GCN
Mode:         VHF Category B, Ground station E
Originator:   CMU (AOC Applications)
Direction:    DWN
UDBI/TecAck:            6/NAK
MesSeqN/BlockSeqN/Suf:  83/A/ETX

Free text:
00067220200 N 47.360/E 18.218/UTC 1502/FOB  105/ALT  35000/CAS 2675
================================
ACARS Message decoded at [ Čt 22. červenec 2010, 19:54:38  UTC ]:
Label:            ACARS link test.
Flight:       TK 1785
Address:      TC-JFD
Mode:         VHF Category B, Ground station E
Originator:   System Control
Direction:    DWN
UDBI/TecAck:            0/NAK
MesSeqN/BlockSeqN/Suf:  51/A/ETX

Free text:

=========== FINISHED ===========
Summary:
     2 messages with both parity and CRC correct.
   287 messages with bad parity, CRC not checked.
     8 messages with good parity but bad CRC.
```

```
 pyacars % sox ../sounds/ACARS_vr5000_ATT_RF128_recout.flac \
-b32 -e float -t raw -r9600 - trim 0 1:04 | \
./coherent.py stdin 9600 -f csv
Using decimation factor 1.
>>> gr_fir_fcc: using SSE
>>> gr_fir_ccf: using SSE
>>> gr_fir_fff: using SSE
utc_year,utc_month,utc_day,utc_hour,utc_minute,utc_second,label,t_label,ali,fn,address,mode,t_mode,originator,t_originator,direction,
ubi_dbi,msn,t_msn,bsc,t_bsc,tech_ack,free_text,t_end,raw
2010,7,22,19,55,21.008749,1L,1L,MS,0732,SU-GCN,E,"VHF Category B, Ground station E",M,CMU (AOC Applications),d,6,83,83,A,A,,00067220200 N
47.360/E 18.218/UTC 1502/FOB  105/ALT  35000/CAS 2675,ETX,E.SU-GCN1L6M83AMS073200067220200 N 47.360/E 18.218/UTC 1502/FOB  105/ALT  35000/CAS
2675
2010,7,22,19,55,21.091412,Q0,   ACARS link test.,TK,1785,TC-JFD,E,"VHF Category B, Ground station E",S,System Control,d,0,51,51,A,A,
,,ETX,E.TC-JFDQ00S51ATK1785
Summary:
     2 messages with both parity and CRC correct.
   268 messages with bad parity, CRC not checked.
    10 messages with good parity but bad CRC.
```

```
pyacars % sox ../sounds/ACARS_vr5000_ATT_RF128_recout.flac \
-b32 -e float -t raw -r9600 - trim 0 1:04 | \
./coherent.py stdin 9600 -f raw
Using decimation factor 1.
>>> gr_fir_fcc: using SSE
>>> gr_fir_ccf: using SSE
>>> gr_fir_fff: using SSE
2010-07-22 19:55:57.191290Z E.SU-GCN1L6M83AMS073200067220200 N 47.360/E 18.218/UTC 1502/FOB  105/ALT  35000/CAS 2675
2010-07-22 19:55:57.292208Z E.TC-JFDQ00S51ATK1785
Summary:
     2 messages with both parity and CRC correct.
   271 messages with bad parity, CRC not checked.
    12 messages with good parity but bad CRC.
```