Workshop: “Decoding Interplanetary Spacecraft”

Dr. Daniel Estévez

14 September 2020

Virtual GRCon20

Introduction

In this workshop:

- We start off with IQ recordings of some deep-space probes
- Use GNU Radio to analyze the signal and find its specs (simple reverse-engineering)
- Build a decoder to obtain telemetry frames as we go

Why?

- It is cool!
- It isn't difficult: probably easier than your average ASK keyfob
- To learn deep-space communications

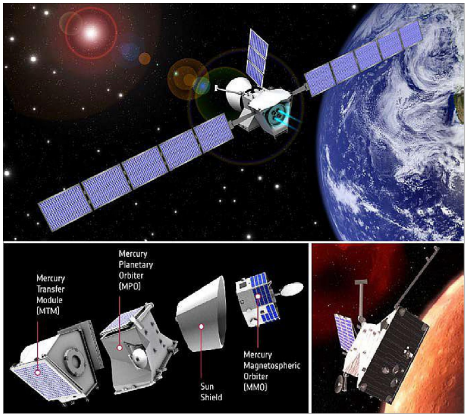
The focus is on hands-on with GNU Radio, but there are some slides with the theoretical background. You can always come back to these later.

ESA Solar Orbiter

ESA mission to study the heliosphere and solar wind (some participation of NASA)
Launched on 10 February 2020

* Image from ESA Science & Exploration

BepiColombo



ESA / JAXA mission to Mercury
Launched on 20 October 2018, Earth flyby on 10 April 2020

* Image from eoPortal Directory

Receiving deep-space probes signals

These spacecraft transmit telemetry in the X band, around 8.4GHz

The signal can be detected with “small” dishes from long distances, but to have enough SNR to demodulate the data, the spacecraft needs to be near Earth

The recordings used here have been made by Paul Marsh M0EYT:

- Solar Orbiter: 3 days after launch, 1.7 million km away
- BepiColombo: 6 days before Earth flyby, 2 million km away

More is possible: Paul is currently demodulating Tianwen-1 at 16 million km on its way to Mars

Put these numbers in perspective: the Earth-Moon distance is 0.4 million km; light takes 10 seconds to travel 3 million km

This is Paul's 2.4m antenna.

Signal spectrum

The first step in analyzing a signal is looking at the spectrum and/or waterfall
This requires computing FFTs to pass from time domain to frequency domain

The *QT GUI Frequency Sink* and *QT GUI Waterfall Sink* GNU Radio blocks can be used for this

inspectrum, by Mike Walters is also a good tool for viewing the waterfall of a recording
`https://github.com/miek/inspectrum`

Hands-on: Visualizing the spectrum

Phase modulation in deep-space communications

Many deep-space comms systems use residual-carrier phase modulation:

$$x(t) = A\cos(2\pi f_0 t + \varphi(t)) = \mathrm{Re}(x_{BB}(t)\exp(2\pi i f_0 t)), \quad x_{BB}(t) = A\exp(i\varphi(t))$$

$$\varphi(t) = Ds(t)d(t), \quad s(t) = h(\sin(2\pi f_{sc} t + \varphi_{sc})), \quad d(t) = \sum_{n\in\mathbb{Z}} d_n p(t/T_s - n),$$

where D is the deviation, $h(t) = t$ for a sine-wave subcarrier, $h(t) = \mathrm{sign}(t)$ for a square-wave subcarrier, $d_n = \pm 1$, and p is the data pulse shaping filter, typically $p(t) = 1$ for $0 \leq t \leq 1$, and $p(t) = 0$ otherwise.

Manchester encoding is a special case, where $h(t) = \mathrm{sign}(t)$, $f_{sc} = 1/T_s$ and $\varphi_{sc} = 0$.

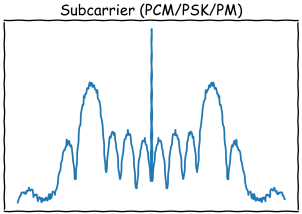
The residual carrier can be used to recover the carrier phase

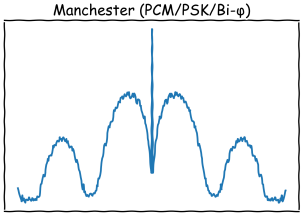
Suppressed-carrier modulations, such as GMSK/OQPSK are also possible, especially for high-speed data

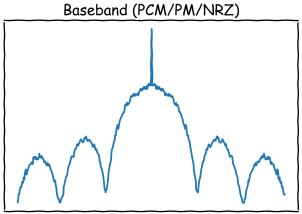
Spectrum depending on the subcarrier

The subcarrier can be used to push the data power away from the suppressed carrier, to help suppressed carrier recovery with a PLL
There are the following possibilities:

- Subcarrier, $f_{sc} > 1/T_s$
- Manchester encoding, $f_{sc} = 1/T_s$
- Baseband modulation, $s(t) = 1$ ($f_{sc} = 0$)







Carrier phase recovery

We use a PLL to recover the phase of the residual carrier. Given

$$y(t) = a(t)e^{i\psi(t)}[1 + z(t)] + n_1(t),$$

with $|a'(t)|$, $|\psi'(t)|$, $|\psi''(t)|$ small, a PLL produces an estimate $\widehat{\psi} \approx \psi$. Here ψ accounts for the transmitter and receiver local oscillator difference and changes in propagation path.

The *PLL Carrier Tracking* GNU Radio block outputs

$$e^{-i\widehat{\psi}(t)}y(t) \approx a(t)[1 + z(t)] + n_2(t).$$

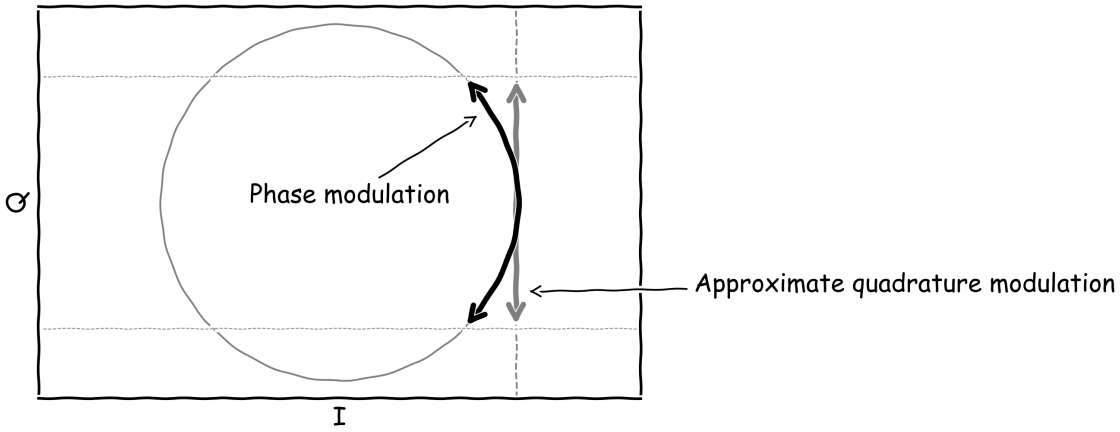
There are other PLL blocks in GNU Radio which output different things

The PLL blocks (and many others) assume a signal amplitude of one ($a(t) \approx 1$), so they are typically used after an AGC. The *RMS AGC* block from gr-satellites estimates the signal RMS power and divides the signal by it.

Hands-on: Carrier phase locking with a PLL

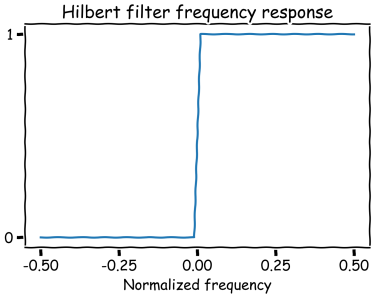
Phase modulation and IQ

Phase modulation can be approximated by a linear modulation where the data is in quadrature with the suppressed carrier, since $\exp(i\varphi(t)) \approx 1 + i\varphi(t)$



Downconversion of the data sideband to baseband

We take the complex argument (or the imaginary part) at the output of the PLL and obtain the real Manchester signal $\varphi(t)$. This has two sidebands. Each of the sidebands looks like BPSK, so we can filter one sideband, move to baseband, and process as BPSK.
We use a Hilbert filter to select only one sideband



Then multiply by a complex exponential to move in frequency. Remember: multiplication by $\exp(2\pi ift)$ shifts the frequency by f.

Hands-on: Downconverting the data sideband to baseband

Finding the carrier frequency of BPSK

A BPSK signal is

$$y(t) = d(t)e^{2\pi ift} + n(t),$$

with $d(t) = \pm 1$. We square it

$$y(t)^2 = e^{4\pi ift} + 2n(t)d(t)e^{2\pi ift} + n(t)^2$$

and obtain a carrier at frequency $2f$.

The same trick can be used for m-PSK, by computing the m-th power.

Hands-on: Refining our downconversion to baseband

Finding the symbol rate

We have a BPSK signal in discrete time

$$y[n] = d[n]e^{i\omega n},$$

with $d[n] = \pm 1$. We compute

$$z[n] = y[n]\overline{y[n-1]} = d[n]d[n-1]e^{i\omega}.$$

Then $z[n] = e^{i\omega}$ except when $n-1$ and n lie in different and opposite symbols, in which case $z[n] = -e^{i\omega}$.

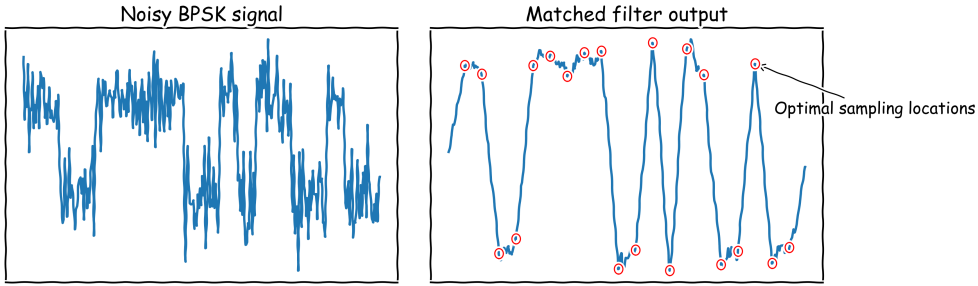
So $z[n]$ is a sequence of pulses that occur on some of the symbol changes of y. The spectrum of z has a strong component at the symbol rate of y.

This trick is a simple example of cyclostationary analysis, which is a very powerful tool.

Hands-on: Finding the symbol rate

Clock recovery

We need to filter the noisy BPSK signal with a matched filter to accumulate the energy of each symbol and reduce noise. For a square pulse, the matched filter is just a moving average. We need to find the optimal sampling locations for the symbols. The *Symbol Sync* GNU Radio block will do that given an estimate of the symbol rate.



Suppressed carrier recovery

After sampling with the *Symbol Sync* block, our symbols look like

$$S[n] = d_n e^{i\psi_n},$$

where $d_n = \pm 1$ and ψ_n changes slowly. We still have a suppressed carrier $e^{i\psi_n}$.

A Costas loop will produce an estimate $\widehat{\psi}_n \approx \psi_n$ and output

$$e^{-i\widehat{\psi}_n} S[n] \approx d_n.$$

There is a 180º phase ambiguity in the Costas loop, so it may happen that $\widehat{\psi}_n \approx \psi_n + \pi$ and we get $-d_n$ instead of d_n.

Note that here we are actually using the Costas loop to recover the Manchester subcarrier, since the carrier was actually recovered by the PLL.

Hands-on: Clock and carrier phase recovery

Coding in deep-space communications: CCSDS

Now that we have symbols, we want to decode frames from them

Space communications protocols are standardized by CCSDS in the Blue Books
The *TM Synchronization and Channel Coding* blue book describes:

- FEC (encode the message to correct bit errors on the receive end)
- Scrambling (make the message look random, for performance)
- Syncwords (mark the beginning of frames)

These are the possible types of FEC:

- Convolutional code (Viterbi decoder)
- Reed-Solomon
- Concatenated (convolutional + Reed-Solomon)
- Turbo code
- LDPC

CCSDS syncwords

The syncword depends on the FEC used. This can help us guess the FEC type.

ASM for uncoded data, convolutional, Reed-Solomon, concatenated, rate-7/8 LDPC for Transfer Frame, and all LDPC with SMTF stream coded data:	`1ACFFC1D`
ASM for rate-1/2 Turbo and rates 1/2, 2/3, and 4/5 Transfer Frame LDPC coded data:	`034776C7272895B0`
ASM for rate-1/3 Turbo coded data:	`25D5C0CE8990F6C9461BF79C`
ASM for rate-1/4 Turbo coded data:	`034776C7272895B0 FCB88938D8D76A4F`
ASM for rate-1/6 Turbo coded data:	`25D5C0CE8990F6C9461BF79C DA2A3F31766F0936B9E40863`

With convolutional codes, the syncword is sent encoded; with the rest, it is sent uncoded

We can try to correlate the symbol stream against these syncwords. Correlation can be implemented by a FIR filter with the *Decimating FIR filter* block.

Hands-on: Finding the syncword

Turbo codeword sizes

We can find the Turbo code parameters from the codeword size, which can be measured from the distance between syncwords

Table 6-2: Codeword Lengths for Supported Code Rates (Measured in Bits)

Information block length k	Codeword length n			
	rate 1/2	rate 1/3	rate 1/4	rate 1/6
1784	3576	5364	7152	10728
3568	7144	10716	14288	21432
7136	14280	21420	28560	42840
8920	17848	26772	35696	53544

We use *Correlate Access Code - Tag* to detect syncwords (add tags) and *Tag debug* to print their locations

Hands-on: Finding the codeword size

Turbo decoding with GNU Radio

We can use the Turbo decoder blocks from gr-dslwp (Longjiang-2 Chinese lunar microsatellite mission) by Wei Mingchuan from Harbin Institute of Technology (China)
This uses an open source Turbo code implementation by Gianluca Marcon, made in 2017 as a course project in Univ. of Padova (Italy)

- gr-dslwp for GNU Radio 3.8
  `https://github.com/daniestevez/gr-dslwp/tree/maint38`
- Gianluca's deepspace-turbo
  `https://github.com/geeanlooca/deepspace-turbo`

We use:

- *Frame Splitter F* to put each codeword in a PDU
- *CCSDS Pseudo Randomizer* to perform descrambling
- *CCSDS Turbo Decode* for actual Turbo decoding

Hands-on: Turbo decoding

Where to go from here? Space Data Link protocols

Frames often use CCSDS protocols: TM Space Data Link or AOS Space Data Link protocols (see respective Blue Books)

Space Data Link frames often have a CRC-16 at the end. We can use it to check that our decoded frames are correct
Exception: Reed-Solomon frames usually don't have a CRC-16, since Reed-Solomon already does error checking

The upper layer protocol is usually CCSDS Space Packets

I typically study these in a Jupyter notebook, but we could also use the CCSDS blocks from gr-satellites, made by Athanasios Theocharis in ESA Summer of Code in Space 2019

Solar Orbiter frames

Solar Orbiter uses TM Space Data Link frames

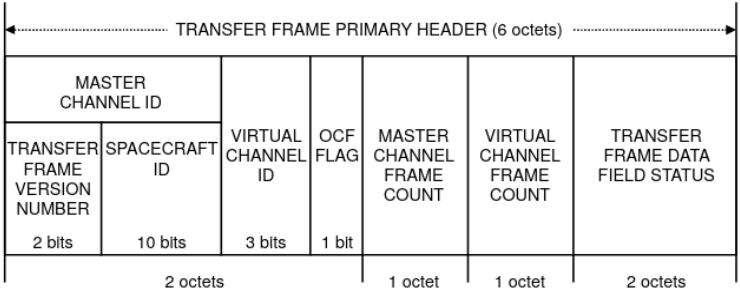


Figure 4-2: Transfer Frame Primary Header

Hint: Both the TM Space Data Link frames and AOS Space Data Link frames start by a Transfer Frame Version Number field. Its value can be used to distinguish them

Hands-on: Brief look at the frames with Python: CRC's and headers

About BepiColombo

BepiColombo's telemetry is very similar to Solar Orbiter

	Solar Orbiter	**BepiColombo**
Modulation	PCM/PM/Bi-φ	PCM/PM/Bi-φ
Baudrate	555.5k	700k
Coding	$r = 1/2$ Turbo	$r = 1/2$ Turbo
Frame size	8920 bits	8920 bits
Fames	TM	TM

The signal in the recording is weaker than Solar Orbiter's, so trying to decode is a good exercise to check decoder performance

Most of the frames contain idle data, but they are interesting to analyze, because they are filled with a PN sequence that resets "unexpectedly"

Further reading

Mars spacecraft (in collaboration with AMSAT-DL and Bochum observatory)
Just search in `http://destevez.net`

	EMM	Tianwen-1	Tianwen-1 (high-speed)
Modulation	PCM/PM/NRZ	PCM/PSK/PM	QPSK
Baudrate	12k	16384	2048k
Coding	$r = 1/6$ Turbo	Concatenated	Concatenated
Frame size	1784 bits	1760 bits	7136 bits
Fames	AOS	AOS	AOS

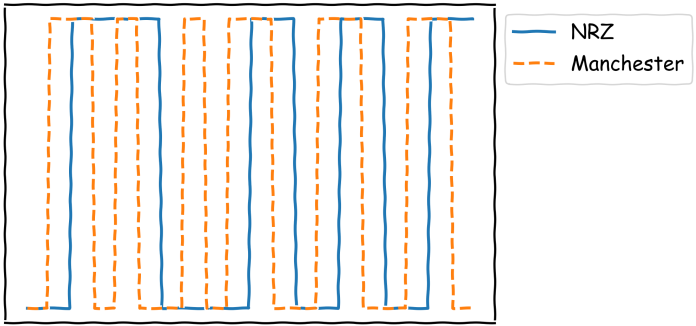
	Mars 2020 (safe mode)	Mars 2020
Modulation	PCM/PSK/PM	PCM/PM/NRZ
Baudrate	80	60k
Coding	$r = 1/2$ Turbo	$r = 1/6$ Turbo
Frame size	1784 bits	8920 bits
Fames	AOS	AOS

Additional material

Two other approaches for Manchester demodulation

We have downconverted a sideband to baseband, which can also be used for PCM/PSK/PM. To demodulate Manchester, we can instead:

- Demodulate as twice the symbol rate, then use *Manchester Sync* from gr-satellites to find and wipe the Manchester clock
- Use an FIR filter as matched filter to the Manchester pulse



Manchester Sync

The *Manchester sync* block gets an input at twice the symbol rate...

$$\cdots 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0\ 1\ 0 \cdots$$

...finds the phase of the Manchester clock, by making pairs that look like `01` and `10`...

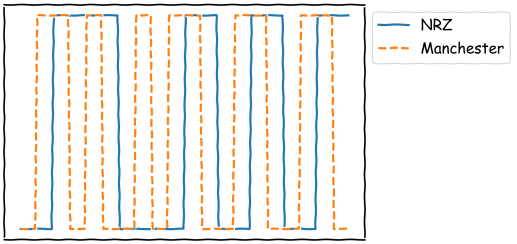
$$\cdots 1|0\ 1|1\ 0|1\ 0|0\ 1|0\ 1|1\ 0|1\ 0|0\ 1|1\ 0|0\ 1|1\ 0|1\ 0 \cdots$$

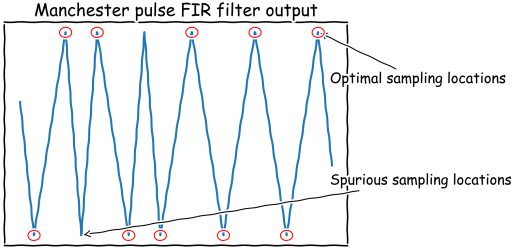
...and “wipes” the Manchester block, by replacing each Manchester pair by the corresponding NRZ symbol...

$$\cdots\ |\ 0\ |\ 1\ |\ 1\ |\ 0\ |\ 0\ |\ 1\ |\ 1\ |\ 0\ |\ 1\ |\ 0\ |\ 1\ |\ 1 \cdots$$

Manchester pulse matched filter

If we use the Manchester pulse `10` as a matched filter, we wipe the Manchester clock and obtain the correct symbols at the optimal sampling locations.
We also obtain spurious sampling locations between adjacent and equal symbols.





However, the clock phase for spurious sampling locations is unstable, so a DTLL (*Symbol Sync*) will tend to lock to the correct optimal sampling locations phase.
It is good to resample to an even samples-per-symbol before the FIR filter.