gr-satellites
A collection of GNU Radio decoders for Amateur satellites

Dr. Daniel Estévez

18 September 2019
GNU Radio Conference 2019, Huntsville (AL)
About the speaker

- PhD in Mathematics
- Day job: in GMV (Madrid) as a GNSS engineer, developing GNSS receivers and simulators, especially for Galileo
- Independent researcher in radio communications, radio science, space systems and other topics
- Amateur radio operator: EA4GPZ (Spanish callsign), M0HXM (UK callsign)
- Blog [http://destevez.net](http://destevez.net)
- Twitter @ea4gpz
Outline

1. Introduction
2. A brief look inside
3. Roadmap
4. Demo time!
Outline

1. Introduction
2. A brief look inside
3. Roadmap
4. Demo time!
A GNU Radio out-of-tree module with a collection of telemetry decoders for Amateur satellites

Input: IQ RF samples (from SDR, conventional radio or recording)
Output: packets in hex or parsed telemetry values
Currently supports more than 80 different satellites
More than 70 custom blocks, many of them implemented in Python
Project goal: providing an open-source solution for decoding every satellite that transmits on Amateur bands
Essentially a one man’s project, but I’m eager to collaborate with other people
Origins of the project

- Started in 2015 as experiments to decode some Amateur satellites which nobody had decoded before (other than the satellite owners). These often involved some reverse-engineering.

- Motivations:
  - Learning and fun
  - ITU Radio Regulations state that Amateur transmissions “shall not be encoded for the purpose of obscuring their meaning”. This means that all Amateur transmissions should have a readily available decoder or public specifications.

- People started to find these experiments useful, so the idea to collect them under a comprehensive collection gave rise to gr-satellites

- Since then, I have been adding support for newer Amateur satellites as they get launched
Usual development workflow

- A new satellite gets launched
- Amateurs do some recordings of the signal
- I work with the recordings and documentation available online to see what protocols/specifications are used
- Usually the documentation is incomplete or inexistent: do reverse-engineering or try to get in touch with the satellite team to ask questions
- If all goes well, eventually we figure out all the specifications
- Write a decoder, put up a blog post
Currently there is an ongoing discussion in the Amateur community about the importance of publishing documentation and specifications about the signals used by Amateur satellites.

We are trying to get things more strict: complete specifications must be publicly available at some point before launch.

If you are designing a satellite that will use Amateur radio spectrum, please do:
- Get in contact with the Amateur community. We are here to help.
- Write and publish good and complete specifications for your protocols.
1 Introduction

2 A brief look inside

3 Roadmap

4 Demo time!
Each satellite has its own flowgraph
Basic information about each flowgraph is included in the README
The flowgraph contains the telemetry decoder (from IQ to PDUs) and telemetry parsers, image decoders and telemetry submitters as appropriate
No GUI
Some configuration parameters. Designed to run as a .py script from the terminal.
Output gets printed to the terminal, or passed on via sockets or files
Supporting different input formats can be cumbersome.
I settled on real time UDP input with a (real) float32 signal at 48ksp.

Reasons:
- Many people have 48ksps audio recordings from conventional radios or SDRs.
- GQRX can stream audio with UDP using this format.

Depending on the modulation, something different is expected:
- FM/FSK. FM demodulated audio.
- Narrow bandwidth linear (e.g. 1k2 BPSK). Conventional SSB audio (0-3kHz) with signal centred at 1.5kHz.
- Wide bandwidth linear (e.g. 9k6 BPSK). Wide SSB audio (0-24kHz) with signal centred at 12kHz.

Ways to feed input:
- GQRX
- gr-frontends: UDP streamers from WAV recordings, audio source, and some SDR hardware.
- Build your own using netcat.
A very useful block: Sync and create (packed) PDU

- Most satellites transmit packets of a fixed size or with a (small) MTU
- Packets are marked by a syncword at the beginning
- Sync and create PDU extracts a PDU of fixed size whenever the syncword is detected
- “Overlapping packets” are allowed. Useful for shorter packets or false syncword detections
Fixed Length Packet Tagger is a custom Python block. It outputs a packet and `packet_len` tag whenever it sees a syncword tag.

Maybe I should be using Protocol Parser for this?
Working with KISS files to store packets

- The KISS protocol was originally designed to interface a TNC (packet radio modem) and a host.
- It is a way of marking frame boundaries and sending control commands.
- These days it is often used to delimit frame boundaries in files, TCP streams, etc.
- Very simple protocol: one frame-delimiter byte to mark frame boundaries, one escape byte to escape the frame-delimiter byte or the escape byte if they occur in the data.
Custom HDLC Framer and Deframer

- Implemented in Python
- They do not have NRZ-I built in. Sometimes HDLC is used without NRZ-I (usually a bad idea).
- Framer can add a preamble and postamble of arbitrary length. Long preamble important for clock recovery in the receiver.
- Deframer can skip CRC-16 check. Useful for debugging.
Some other useful components

- Decoders for GOMspace radios U482C and AX100. Many satellites use these.
- Texas Instruments CC11xx and SiLabs Si4463 decoders.
- Several FEC decoders:
  - CCSDS Viterbi
  - CCSDS or general Reed Solomon (uses Phil Karn’s libfec)
  - Reed Solomon decoder with rscode (perhaps redundant)
  - BCH decoder
  - Golay decoder
- Several descramblers:
  - G3RUH asynchronous
  - CCSDS synchronous
  - IESS-308 asynchronous
  - PN9 synchronous (TI and SiLabs variants)
- Several CRC checkers
1. Introduction

2. A brief look inside

3. Roadmap

4. Demo time!
June. Contract with SatRevolution to adapt gr-satellites to decode their Świątowid and KRAKsat satellites

July–September. ESA Summer of Code in Space: Athanasios Theocharis (Univ. of Thessaloniki) adding blocks for the CCSDS Space Packet, TM Space Data Link and TC Space Data Link protocols
On the roadmap

- Port to GNU Radio 3.8 and Python 3
- Rearchitecture:
  - More modularity in the decoders: easier to add a satellite, easier to change some component in all applicable satellites
  - Flexibility in selecting the outputs: selecting different kinds of outputs, ability to output different things to files
  - Flexibility in selecting the inputs: UDP IQ realtime input, WAV file at faster speed
  - Perhaps optional GUI elements
  - What other features would be useful?
- Including FSK demodulators by David Rowe
- Adding tests. End-to-end tests with sample recordings from satellite-recordings.
- Not yet 100% sure on how to go about these, so comments are welcome
- Remember that gr-satellites is my largest project, but it is also a kind of “side project”. It only gets perhaps 10% of my time. Remaining 90% goes to other varied smaller projects or experiments.
Outline

1. Introduction
2. A brief look inside
3. Roadmap
4. Demo time!