Open Research Institute
Inner Circle Newsletter
June 2025

Want more Inner Circle Newsletters?
Use the QR code at left or go to
http://eepurl.com/h_hYzL
and sign up.

Icom IC-905 10 GHz Polarization Issue	page 2
Open Source Cubesat Workshop 2025 Announced	page 4
Take This Job	page 5
Opulent Voice Protocol Development	page 6
Efficiently Using Transmitted Symbol Energy via Delay Doppler Channels pt 2	page 8
Wireshark Plugin for Opulent Voice	page 18
Guide to Transmitting DVB-S2 Video using ORI Encoder	page 18
Postlocutor, a Prototype Receiver for Opulent Voice	page 18
Inner Circle Sphere of Activity	back page

The Who What When Where Why

Open Research Institute is a non-profit dedicated to open source digital radio work on the amateur bands. We do both technical and regulatory work. Our designs are intended for both space and terrestrial deployment. We're all volunteer and we work to use and protect the amtaeur radio bands. You can get involved in our work by visiting https://openresearch.institute/getting-started

Membership is free. All work is published to the general public at no cost. Our work can be reviewed and designs downloaded at https://github.com/OpenResearchInstitute

We equally value ethical behavior and over-the-air demonstrations of innovative and relevant open source solutions. We offer remotely accessible lab benches for microwave band radio hardware and software development. We host meetups and events at least once a week. Members come from around the world.

Icom IC-905 10 GHz Polarization Issue

Good news for amateur radio operators using the Icom IC-905 10 GHz system!

Following feedback from the San Bernardino Microwave Society (SBMS), Gordon West WB6NOA, and other users, Icom America has agreed to address the polarization configuration issue that's been affecting signal performance.

The issue: The IC-905's dish antenna comes configured for vertical polarization. Vertical polarization is standard in Japan. However, most U.S. amateur microwave operations use horizontal polarization.

This mismatch has resulted in significant signal strength differences during contacts.

The solution is simple: rotate either the entire dish or just the feed horn 90 degrees to achieve horizontal polarization. However, many operators weren't aware of this requirement.

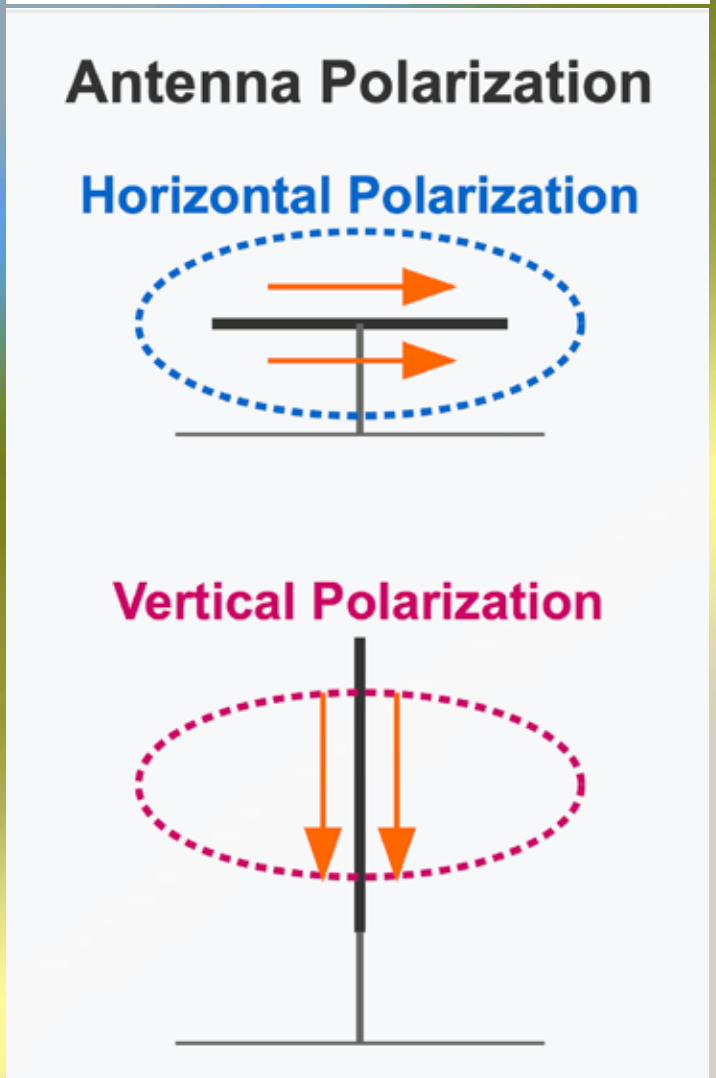
Ray Novak (N9JA) from Icom America has committed to updating the product instructions and adding a label to the packaging that alerts customers about the dish's default vertical polarization and how to configure it for horizontal operation.

Icom has also expressed willingness to work with SBMS on developing educational content about this issue, potentially including information about SBMS membership.

Amateur radio groups interested in contributing to this effort should contact Ray Novak directly at raynovak@icomamerica.com.

This collaborative approach between manufacturers and the amateur radio community demonstrates how user feedback can lead to improved documentation and better user experiences for microwave enthusiasts.

Below is a diagram showing the 90 degree relationship between horizontal and vertical polarization. Rotating the dish or feed by 90 degrees restores the 20-30 dB loss that you will get if your equipment is cross-polarized.



The Microwave Link Mystery

Four amateur radio operators (Alice, Bob, Carol, and Dave) are setting up 10 GHz microwave links. Each has a different antenna polarization: Horizontal, Vertical, Right-Hand Circular (RHC), and Left-Hand Circular (LHC).

Polarization Loss Rules (line of sight)

- Same polarization: 0 dB loss
- Cross-polarized linear (H vs V): 20+ dB loss
- Circular to linear: 3 dB loss (either direction)
- Opposite circular (RHC vs LHC): 20+ dB loss

1. Alice can communicate with Bob with perfect signal strength (0 dB loss).

2. Alice gets terrible reception from Carol (20+ dB loss).

3. Alice receives Dave's signal at reduced power (3 dB loss).

4. Bob can barely hear Carol (20+ dB loss).

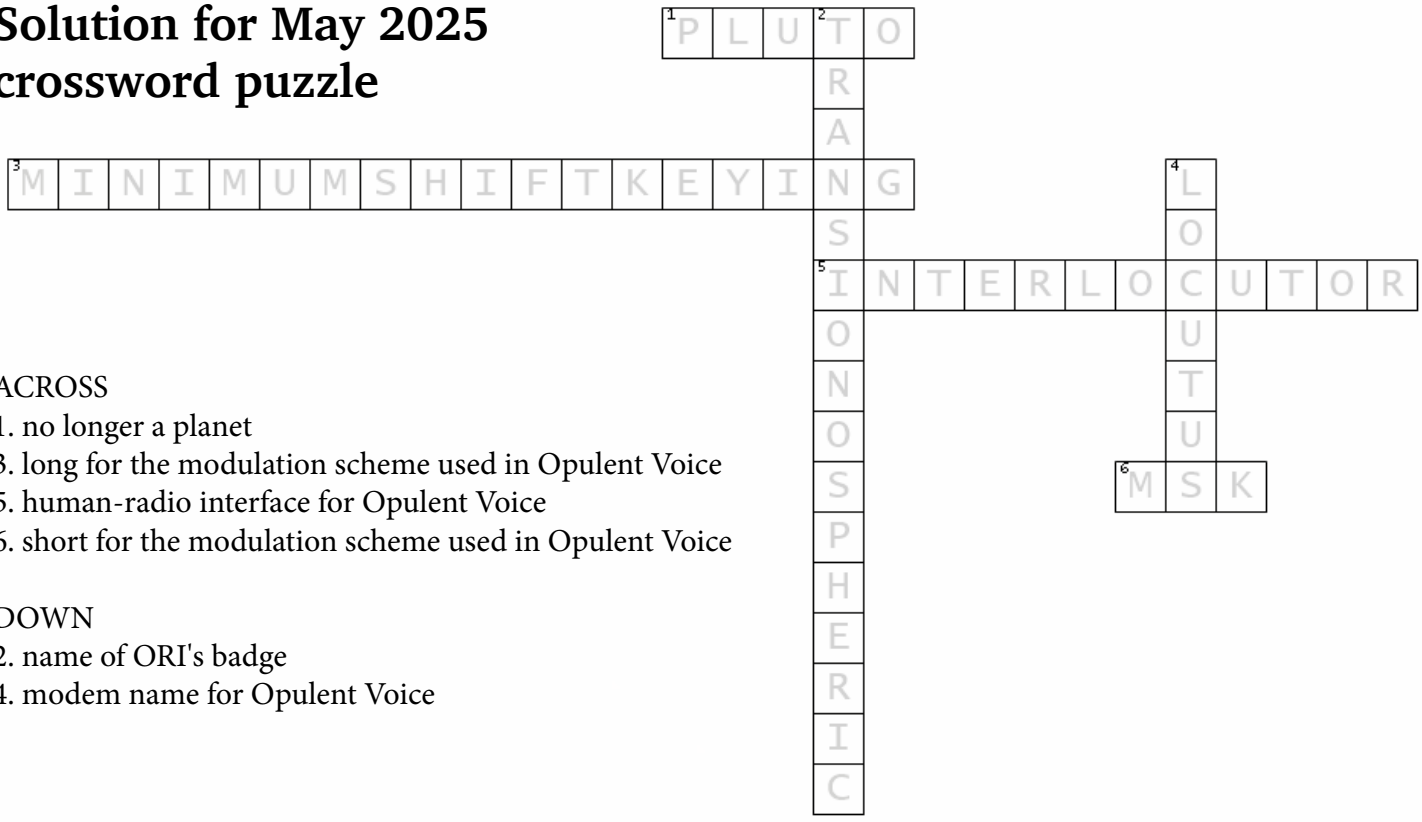
5. Bob gets a good but reduced signal from Dave (3 dB loss).

6. Carol receives Dave's signal at reduced power (3 dB loss).

7. One operator forgot to rotate their new IC-905 dish from its factory vertical polarization setting.

Who has which antenna polarization?

Solution for May 2025 crossword puzzle



ACROSS
1. no longer a planet
3. long for the modulation scheme used in Opulent Voice
5. human-radio interface for Opulent Voice
6. short for the modulation scheme used in Opulent Voice

DOWN
2. name of ORI's badge
4. modem name for Opulent Voice

Open Source Cubesat Workshop 2025 Announced

https://events.libre.space/event/9/overview

Want to represent ORI? Join our Slack workspace and participate in the process of applying to present a talk, workshop, or roundtable. Information from Libre Space Foundation about the workshop is below:

Reignite the Open Source CubeSat Workshop!

Join Libre Space Foundation again for a two-day workshop to see how the open source approach can be applied to CubeSat missions with a focus on innovative and state-of-the-art concepts!

Open source software and hardware is empowering and democratizing all areas of life, so why not apply it to space exploration? The Open Source Cubesat Workshop was created exactly for that: to promote the open source philosophy for CubeSat missions and beyond. The sixth edition of the workshop takes place in Athens, Greece, hosted by Libre Space Foundation.

CubeSats have proven to be an ideal tool for exploring new ways of doing space missions; therefore, let’s remove the barrier of confidentiality and secrecy, and start to freely share knowledge and information about how to build and operate CubeSats. This workshop provides a forum for CubeSat developers and CubeSat mission operators to meet and join forces on open source projects to benefit from transparency, inclusivity, adaptability, collaboration and community.

The focus of this year’s workshop is to develop and apply open source technologies for all aspects of a space mission. The target audience of this workshop is academia, research institutes, companies, and individuals.

Starts Oct 25, 2025, 10:00 AM
Ends Oct 26, 2025, 6:00 PM

Europe/Athens
Serafio of the Municipality of Athens

19 Echelidon Street & 144 Piraeus Street 11854, Athens Greece

The workshop is free of charge but limited to 200 people.

“Take This Job”

30 June 2025

Interested in Open Source software and hardware? Not sure how to get started? Here’s some places to begin at Open Research Institute. If you would like to take on one of these tasks, please write hello@openresearch.institute and let us know which one. We will onboard you onto the team and get you started.

Opulent Voice:

- Add a carrier sync lock detector in VHDL. After the receiver has successfully synchronized to the carrier, a signal needs to be presented to the application layer that indicates success. Work output is tested VHDL code.
- Bit Error Rate (BER) waterfall curves for Additive White Gaussian Noise (AWGN) channel.
- Bit Error Rate (BER) waterfall curves for Doppler shift.
- Bit Error Rate (BER) waterfall curves for other channels and impairments.
- Review Proportional-Integral Gain design document and provide feedback for improvement.
- Generate and write a pull request to include a Numerically Controlled Oscillator (NCO) design document for the repository located at https://github.com/OpenResearchInstitute/nco.
- Generate and write a pull request to include a Pseudo Random Binary Sequence (PRBS) design document for the repository located at https://github.com/OpenResearchInstitute/prbs.
- Generate and write a pull request to include a Minimum Shift Keying (MSK) Demodulator design document for the repository located at https://github.com/OpenResearchInstitute/msk_demodulator
- Generate and write a pull request to include a Minimum Shift Keying (MSK) Modulator design document for the repository located at https://github.com/OpenResearchInstitute/msk_modulator
- Evaluate loop stability with unscrambled data sequences of zeros or ones.
- Determine and implement Eb/N0/SNR/EVM measurement. Work product is tested VHDL code.
- Review implementation of Tx I/Q outputs to support mirror image cancellation at RF.

Haifuraiya:

- HTML5 radio interface requirements, specifications, and prototype. This is the primary user interface for the satellite downlink, which is DVB-S2/X and contains all of the uplink Opulent Voice channel data. Using HTML5 allows any device with a browser and enough processor to provide a useful user interface. What should that interface look like? What functions should be prioritized and provided? A paper and/or slide presentation would be the work product of this project.
- Default digital downlink requirements and specifications. This specifies what is transmitted on the downlink when no user data is present. Think of this as a modern test pattern, to help operators set up their stations quickly and efficiently. The data might rotate through all the modulation and coding, transmitting a short loop of known data. This would allow a receiver to calibrate their receiver performance against the modulation and coding signal to noise ratio (SNR) slope. A paper and/or slide presentation would be the work product of this project.

Opulent Voice Protocol Development

Opulent Voice is an open source high bitrate digital voice (and chat, and data!) protocol developed by ORI. It's designed as the native digital uplink protocol for ORI's broadband microwave digital satellite transponder project. Opulent Voice (OPV) is good for both space and terrestrial use.

Locutus

The focus of most of the recent work has been on the minimum shift keying (MSK) modem, called Locutus. The target hardware for implementation is the PLUTO SDR from Analog Devices. See https://github.com/OpenResearchInstitute/pluto_msk for source code, documentation, and installation instructions.

The modem in an excellent "rough draft" state. There are positive results in over-the-air testing.

Interlocutor: The Human-Radio Interface

Interlocutor is the part of the design that takes input audio, text, and keyboard chat from the operator and processes this data into frames for the modem.

Current Interlocutor code can be found at https://github.com/OpenResearchInstitute/interlocutor

Interlocutor also handles received audio, text, and data. The target hardware for this part of the design is a Raspberry Pi version 4, with expectations that it will run on other Python-capable and/or Linux-based devices.

What's Happened Since Our May 2025 Updates?

What were the "Next Steps" from last month? We intended to take the functional Opulent Voice framing, which successfully delivered Opus packets, and insert COBS, RTP, UDP, and IP layers in order to gain a substantial increase in functionality.
All of that has happened. The stream of Opulent Voice frames are much more functional and can be handled by a very large number of existing applications. The additional overhead was well worth the investment of complexity and latency.

What Else is New?

We implemented *configuration files*. This is a YAML file that stores settings for the radio.

A *configuration manager* codebase was added to handle config file tasks.

We implemented *audio hardware configuration files*. This is a YAML file that stores settings for the audio hardware choices made by the operator.

We implemented an *audio hardware manager*. The audio hardware manager helps the operator test, list, and save audio hardware configurations. A microphone gain measurement and speaker test are included.

We wrote an *automated test suite.* It has four phases, with multiple tests per phase. As development proceeds, we can easily test if any code changes damage previously achieved functionality. If it's in the test suite, then it needs to still be working regardless of anything we add to the codebase. This helps ensure a quality product moving forward.

Automated Test Suite Overview

Phase 1: Basic Operations
```
run_test "help_display"
run_test "version_info"
run_test "no_callsign"
run_test "invalid_callsign"
```

Phase 2: Command Line Options
```
run_test "audio_help_options"
run_test "list_audio_exits_clean"
run_test "test_audio_exits_clean"
run_test "setup_audio_handles_eof"
```

```
Phase 3: Network Configuration
    run_test "custom_ip"
    run_test "custom_port"
    run_test "invalid_ip_accepted"
    run_test "invalid_port"

Phase 4: Operating Modes
    run_test "chat_only_mode"
    run_test "verbose_mode"
    run_test "quiet_mode"

Phase 5: Configuration File Handling
    run_test "no_config_files"
    run_test "partial_config"
    run_test "corrupted_config"
    run_test "good_config_loaded"
    run_test "cli_overrides_config"
```

Next Steps?

If the Opus encoder does a good enough job of detecting when the speaker is silent, we can take advantage of those silence frames to send (fragments of) waiting control, text, or data packets. This would be useful mainly if we are also multiplexing multiple streams with an advanced variant of COBS.

We have an ongoing discussion on how to deal with the potential problem of getting out-of-order frames, which may happen when we use Interlocutor with a remote computer network, or with a modem that is (significantly) remote.

An out-of-order Opulent Voice frame problem isn't expected to occur over the radio link. Packets either get through or they don't, but never out of order, because they are always transmitted in order and there's no mechanism to retry transmitting them later. This assumes that the modem is located close to Interlocutor.

So the problem can only occur if we try to use this radio link protocol as a network (transport) protocol, which can introduce complications like out of order delivery. This happens when we're communicating to another computer with Interlocutor software, over the Internet. Which, we expect people might want to do.

The obvious solution here is to use TCP, which is designed as a transport protocol. TCP can turn a channel that delivers packets unreliably and possibly out of order into a channel that never does either of those things. Bytes always arrive perfectly correct and perfectly in order, or else the link fails.

So where we are using just UDP to encapsulate our OPV frames on a short wire between the host and the modem, we would probably want to use TCP on a network link if we for some reason wanted to remote the modem over a network.

Since we already have a configuration item for "computer" or "modem", then we have a way to distinguish between the two modes. The difference between the two modes right now in the code is that selecting "computer" means a KEEPALIVE signal is sent as a control channel message. The KEEPALIVE is not sent when Interlocutor is configured as being connected to a "modem". We could use TCP in computer and UDP in modem. Discussions on this will continue!

Voice detection or VOX? Live voice-to-text transcripts? Smart QSO logging?

Want to Help?

If you would like to help make these and the many other things that we do happen more quickly or better, then you are welcome at ORI!

Read over our code of conduct and developer and participant policies on our website at https://www.openresearch.institute/developer-and-participant-policies/, and then visit our getting started page at https://www.openresearch.institute/getting-started/

Efficiently Using Transmitted Symbol Energy via Delay-Doppler Channels — Part II
Pete Wyckoff, KA3WCA

Introduction

Part II builds examples with Delay-Doppler channels from the Part I foundation of multi-path with delay but without Doppler. Part I defined an efficiency metric as the ratio of energy-per-symbol that actually reaches the demodulator versus the energy-per-symbol transmitted [1]. The efficiency metric showed that certain DSP waveform designs transfer energy from the transmitter to the demodulator more efficiently than others when there is multi-path propagation and a known channel response [2]. This effect exploits superposition of fields from two or more propagation paths. Figure 1 illustrates the effect at a receiver's low intermediate frequency (IF) to simplify visualization.

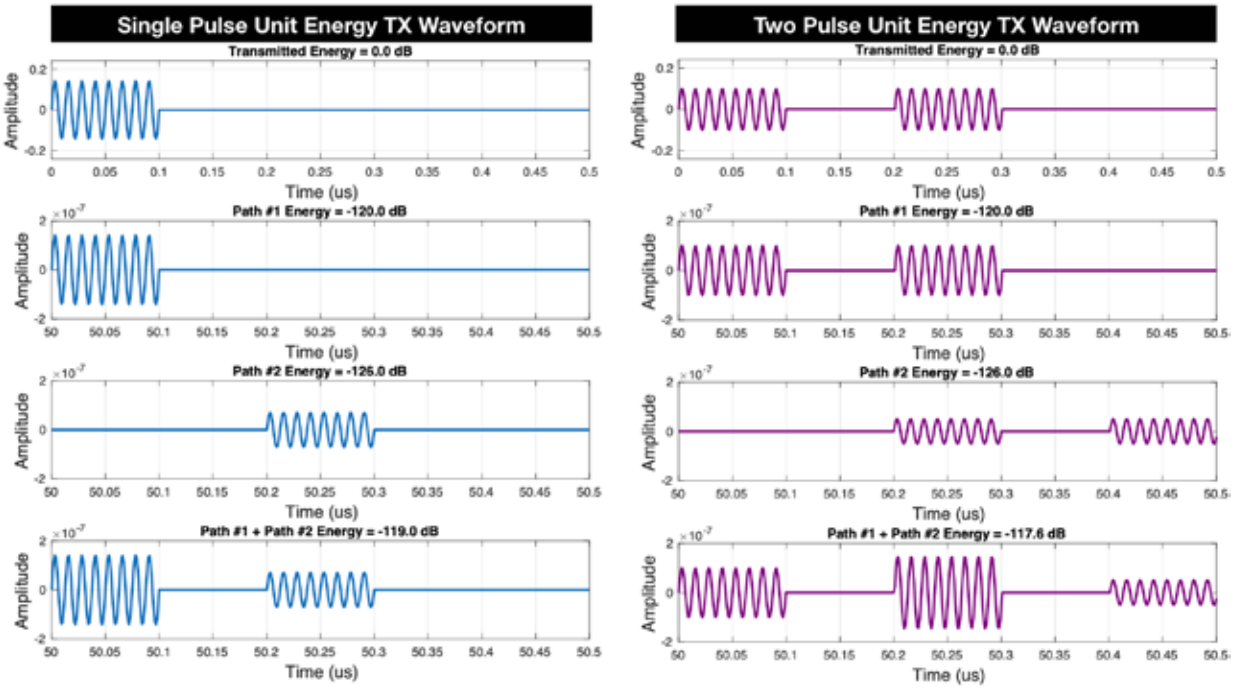


FIGURE 1: (LEFT) A single pulse unit energy transmit waveform propagates to the receiver via two paths and delivers energy -119.0 dB to the receiver. (RIGHT) A two pulse unit energy transmit waveform propagates via the same two paths and delivers energy -117.6 dB to the receiver — *boosting energy transfer from transmitter to receiver by +1.4 dB for this channel*.

The Figure 1 top row shows two alternatives for the transmitted waveform design — *single pulse vs. two pulse waveform*. Crucially, both waveforms have equal transmitted energy-per-symbol. The second row in Figure 1 shows each waveform after propagating via the first path. The third row of Figure 1 shows each waveform after propagating via the second path featuring somewhat greater path delay and further decreased amplitude.

The energy received from each individual path is not affected by using one pulse or two pulses for the transmitted waveform in Figure 1. Rather, superposition of these two paths at the receiver antenna produces the important effect — *see bottom row of Figure 1*. Fields from the two paths combine constructively for the two pulse signal between times 50.2 and 50.3 us. This boosts the overall received energy for the two pulse waveform. In Figure 1, the two pulse waveform transferred energy from the transmitter to the receiver +1.4 dB more efficiently than

the single pulse waveform for this channel. Of course, this assumes we have an accurate estimate of the channel and the boost in efficiency would degrade with errors in that estimate.

Some amateur radio channels exhibit not only multi-path with delay, but multi-path with Delay-Doppler. Figure 2 repeats the same experiment with different Doppler on each path. This shows the two pulse waveform design no longer delivers any boost in energy transferred. Both signal designs deliver -119 dB of energy. The reason is that superposition of the two paths no longer fully and constructively interferes for the two pulse waveform between 50.2 and 50.3 us.

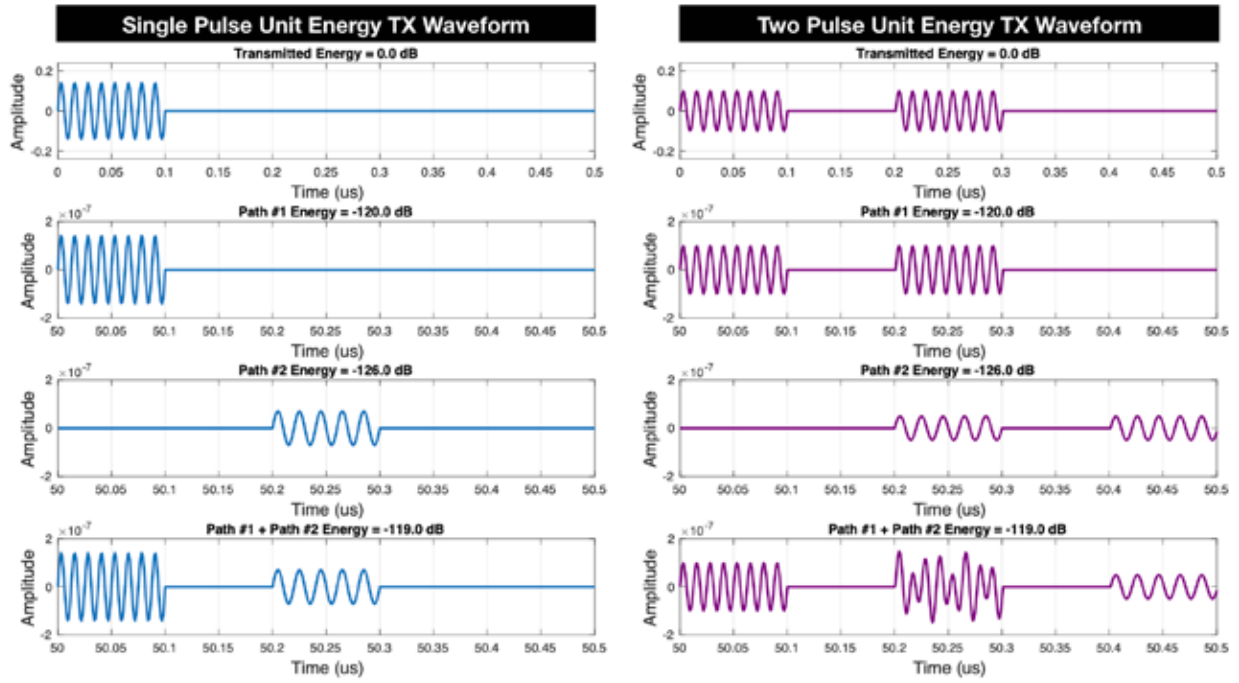


FIGURE 2: Different Doppler on the second path destroys the energy efficiency boost for the two-pulse waveform. Doppler is exaggerated in this diagram to help visualization. Realistic Doppler is addressed later in this paper along with more sophisticated waveform designs.

Improved waveforms to accommodate delay-Doppler channels are the focus of this Part II paper. A simple example resolves the limitation revealed in Figure 2. Then, this foundation is extended to more representative amateur radio channels where Doppler shifts between the different paths might vary by only several Hz over less than a milli-second. Conventional Orthogonal Time-Frequency Space (OTFS) provides inspiration [3]. A new extension of OTFS provides one design methodology to boost the efficiency of energy transfer from transmitter to receiver for such amateur delay-Doppler channels. Finally, performance is characterized with channel estimation errors of varying severity.

Simple Example to Resolve the Doppler Limitation

Different Doppler on the second path will produce fully constructive interference if the two pulses are sent at different center frequencies — *see Figure 3*. This enhanced design restores the two-pulse waveform energy at the receiver to -117.6 dB. The enhanced two pulse waveform is then 1.4 dB more efficient than the single pulse waveform. Incorporating the channel delay and Doppler into the waveform design may increase the energy transferred from the transmitter to the receiver via the delay-Doppler multi-path channel as demonstrated in Figure 3. As in Part I, a waveform with more than two pulses could also be designed to boost efficiency further [4]. Figure 3 merely shows the basic concept for the simplest channel.

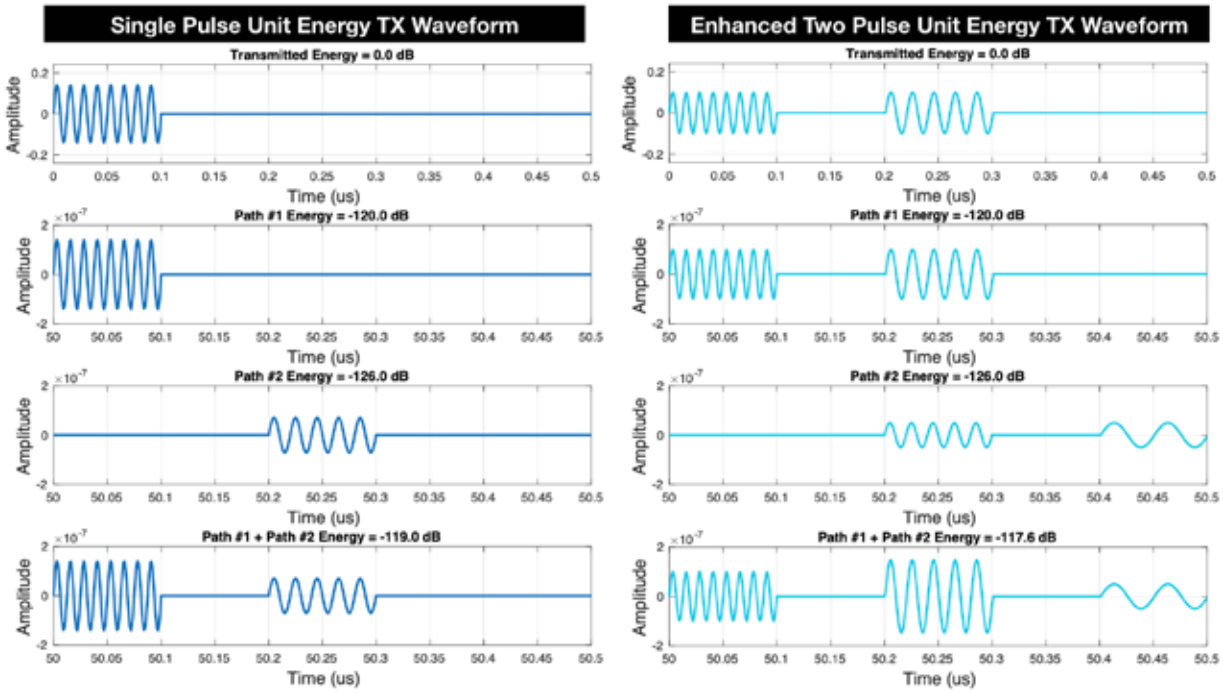


FIGURE 3: The enhanced two-pulse waveform design takes advantage of this known delay-Doppler channel. Transmitting two pulses at different times and frequencies restores the +1.4 dB boost in energy transfer efficiency from the transmitter to the receiver. Doppler is exaggerated in this diagram to help visualization. Realistic Doppler is addressed later in this paper along with more sophisticated waveform designs.

Time-Frequency Perspective on the Two Pulse Waveform & Delay-Doppler Channel

Figure 4 shows a two pulse signal in the time-domain and in the time-frequency domain. The top row shows the transmitted signal from both viewpoints. There are two pulses at the same frequency and separated by 0.2 micro-seconds in time.

The second row in Figure 4 shows the signal that arrives from the first path. It is the transmitted signal shifted in time and reduced in amplitude by 120 dB.

The second path signal appears in the third row of Figure 4. This path is delayed further and reduced in amplitude further. As well, this second path has a Doppler shift with respect to the first path. The time-frequency plot shows this clearly as it resolves the frequency versus time.

The bottom row of Figure 4 shows the superposition of the first and second path in the time domain (lower left) and in the time-frequency domain (lower right). Superposition between 50.2 and 50.3 micro-seconds does not fully constructively interfere because the two paths arrive on two distinct frequencies in Figure 4.

Figure 5 shows the enhanced two pulse signal in time and in time-frequency. The top row of Figure 5 emphasizes the enhanced two pulse signal sends pulses on two different frequencies. The middle two rows show these pulses after traversing the first and second paths. Finally, the bottom row shows this enhanced signal delivers time and frequency alignment of pulses received over the paths from 50.2 to 50.3 micro-seconds.

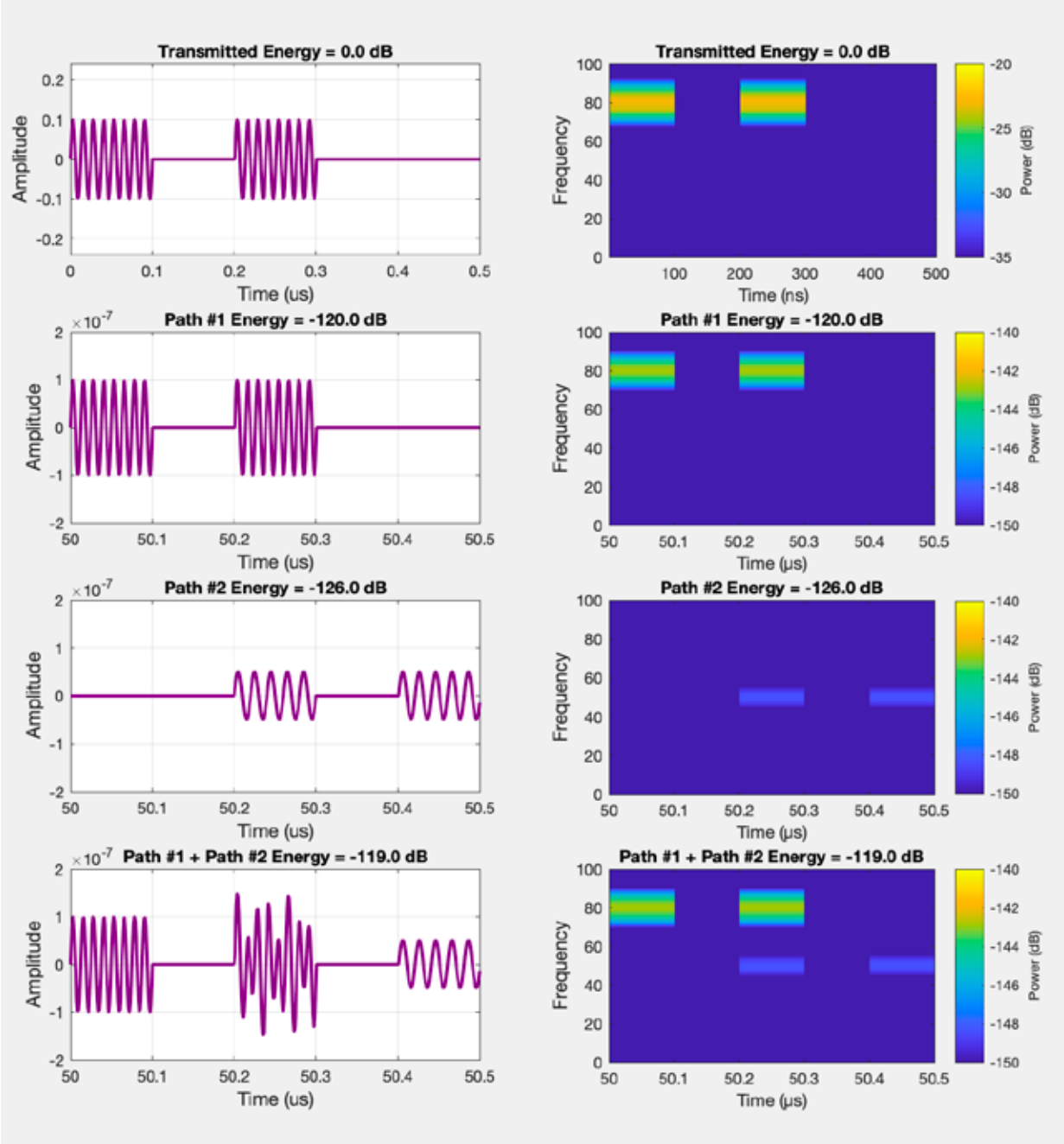


FIGURE 4: Time-frequency perspective shows this transmit waveform design misaligns the pulses in frequency between 50.2 and 50.3 microseconds. There is no boost in energy transfer efficiency since the super-position is not aligned in time and frequency.

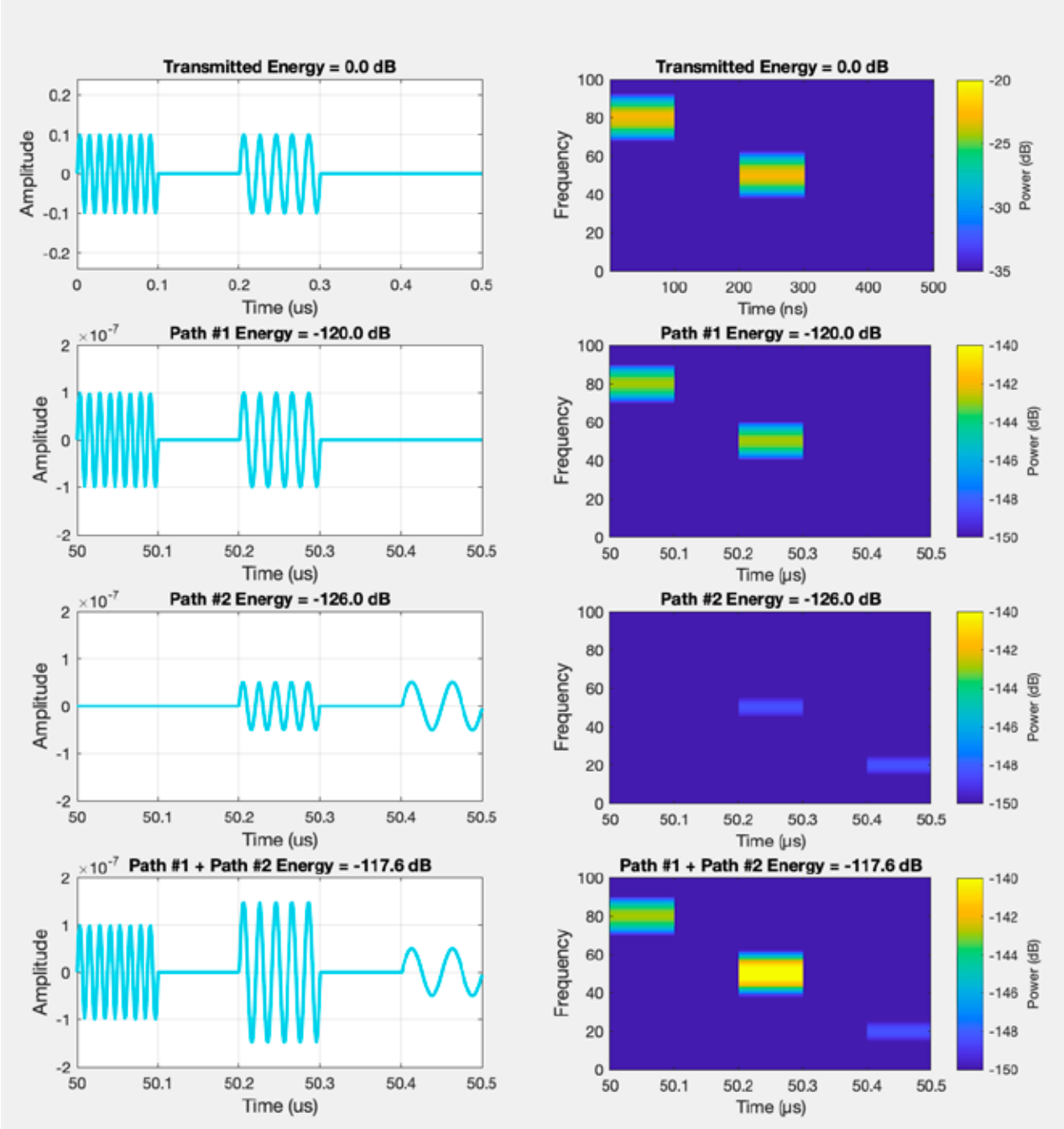


FIGURE 5: Time-frequency perspective shows this transmit waveform design causes a boost in energy delivered to the receiver. The two paths align perfectly between 50.2 and 50.3 micro-seconds, which boosts the received energy.

Extending OTFS Waveforms for More Practical Delay-Doppler Values

This section extends a modulation called orthogonal time frequency space (OTFS) to boost energy efficiency through the multi-path channel. Practically speaking, this is needed because earlier figures exaggerated the Doppler shift between paths and assumed a short symbol period simply to clarify the visualization. That also made the waveform design easier.

More practical constraints require a different approach. Instead of working in time-frequency space, the delay-Doppler space provides a better way. It is only slightly more complicated to boost channel energy transfer efficiency in this manner. The function "*makeOTFS*" creates two series of uniformly spaced pulses that are root-raised cosine filtered and then shifted by a specified Doppler frequency:

```
function tx = makeOTFS(tau, fd, a, Fs, Tsym)
%-------------------------------------------------------------------------
% tx = makeOTFS(tau, fd, Fs, Tsym)
%
% Creates an enhanced OTFS signal designed specifically for the 2-path
% channel.
%
% Variable  Size     I/O  Description
% --------  -------  ---  ---------------------------------------
% tau       1x2      In   Delay for each of two paths in (s)
% fd        1x2      In   Doppler for each of two paths in (Hz)
% a         1x2      In   Complex coef. for each of two paths
% Fs        1        In   Sampling rate in (samples/s)
% Tsym      1        In   Symbol Duration in (s)
% tx        Varies   Out  Transmit signal samples as time-series
%
%                                        -Pete Wyckoff, Version 06172025.
%-------------------------------------------------------------------------
  Tp = max(tau) - min(tau);        %min. time between all path pairs (s)
  T = 2*Tp;                        %prototype pulse period (s)

  for m=1:2                             %two propagation paths
    t = tau(m):T:Tsym;                  %pulse times for mth train (s)
    idx = round(t * Fs) + 1;            %pulse times for mth train (sample)
    x(m,idx) = exp(1i*2*pi*fd(m)*t);    %pulses for mth path
    x(m,idx) = x(m, idx) * exp(-1i*angle(a(m)));  %phase adj.
  end

  [num, den] = rcosine(1/Tp, Fs, 'sqrt', 0.4); %pulse filter
  tx = filter(num, den, sum(x));               %sum pulses & filter
  tx = tx / norm(tx, 2);                       %make unit energy
end %make OTFS
```

Suppose we use the same path coefficients as before, but this time the second path is shifted by merely 0.1 milli-seconds delay and 2 Hz Doppler with respect to the first path:

```
%-------------------------------------------------------------------------
% STEP 1:  define channel paths for delay-Doppler multi-path channel
%-------------------------------------------------------------------------
tau = [0    0.1E-3];                    %delay in (s)
fd =  [0    2];                         %Doppler in (Hz)
a =   [1E-6 0.5E-6];                    %path coefficients (complex)
```

As an experiment, create the enhanced OTFS waveform, pass it through the channel, and report the signal energy as follows:

```
%-------------------------------------------------------------------------
%  STEP 2:  Create OTFS Signal, Pass Through Channel, & Measure Energy
%-------------------------------------------------------------------------
Fs = 1E6;                                   %sampling rate (samples/s)
Tsym = 2;                                   %transmission duration (s)
tx = makeOTFS(tau, fd, a, Fs, Tsym);        %build enhanced OTFS waveform
tx = fliplr(conj(tx));                      %time-reversed & freq. reversed
rx = channelModel(tau, fd, a, Fs, tx);      %pass thru channel
reportEnergy(tx, rx, 'OTFS');               %report results
```

The channel model applies delays, Doppler, magnitude adjustment, and phase rotations:

```
function y = channelModel(tau, fd, a, Fs, x)
%-------------------------------------------------------------------------
% y = channelModel(tau, fd, a, Fs, x)
%
% Propagates input signal through a model 2-path multi-path channel
%
% Variable  Size     I/O  Description
% --------  -------  ---  ---------------------------------------
% tau       1x2      In   Delay for each of two paths in (s)
% fd        1x2      In   Doppler for each of two paths in (Hz)
% a         1x2      In   Path coefficient for each of two paths
% Fs        1        In   Sampling rate in (samples/s)
% x         Varies   In   Row vector of channel input time-series
% y         Varies   Out  Row vector of channel output time-series
%
%                                       -Pete Wyckoff, Version 06172025.
%-------------------------------------------------------------------------
  hLen = round(max(tau)*Fs) + 2;                %length for impulse resp.
  for m=1:2                                     %loop through 2 paths
    h = zeros(1, hLen);                         %initialize as all zeros
    h(round(tau(m)*Fs) + 1) = a(m);             %path delay & coefficient
    p = conv(x, h);                             %apply delay & coef.
    lo = exp(1i*2*pi*fd(m)*(0:length(p)-1)/Fs); %path Doppler
    y(m,:) = p .* lo;                           %apply path Doppler
  end
  y = sum(y);                                   %sum the 2 paths
end %channelModel
```

The function “reportEnergy” computes the energy a prints a message to the terminal window:

```
function E = reportEnergy(tx, rx, idStr)
  txEnergy_dB = 10*log10(sum(abs(tx).^2));
  rxEnergy_dB = 10*log10(sum(abs(rx).^2));
  if(nargout == 0)
    fprintf('\n%s Transmitted Energy = %.1f (dB)', idStr, txEnergy_dB);
    fprintf('\n%s Received Energy = %.1f (dB) \n', idStr, rxEnergy_dB);
  end
  E = [txEnergy_dB, rxEnergy_dB];
end %reportEnergy
```

As a control group, suppose we test the same channel using a PN sequence waveform using similar time and bandwidth:

```
%--------------------------------------------------------------------------
%  STEP 3:  Create PN Signal, Pass Through Channel, & Measure Energy
%--------------------------------------------------------------------------
Fchip = 10E3;                               %chip rate (chips/s)
tx = makePN(Fchip, Fs, Tsym);               %create a PN signal
tx = fliplr(conj(tx));                      %time-reversed & freq. reversed
rx = channelModel(tau, fd, a, Fs, tx);      %pass thru channel
reportEnergy(tx, rx, 'PN');                 %report results
```

The function “makePN” produces this signal as follows:

```
function tx = makePN(Fchip, Fs, Tsym)
%--------------------------------------------------------------------------
% tx = makePN(Fchip, Fs, Tsym)
%
% Creates a root-raised cosine PN sequence waveform
%
% Variable  Size     I/O  Description
% --------  -------  ---  ------------------------------------------------
% Fchip     1        In   Chip rate in (chips/s)
% Fs        1        In   Samp. rate (samples/s) [Integer mult of Fchip!]
% Tsym      1        In   Symbol duration in (s)
% tx        Varies   Out  Row vector of transmit signal as time-series
%
%                                       -Pete Wyckoff, Version 06172025.
%--------------------------------------------------------------------------
  chips = sign(randn(1, round(Tsym*Fchip)));
  [num, den] = rcosine(Fchip, Fs, 'sqrt', 0.4);      %design filter
  tx = filter(num, den, upsample(chips, Fs/Fchip));  %apply filter
  tx = tx / norm(tx, 2);                             %make unit energy
end %makePN
```

The overall script reports the following to the terminal window:

```
OTFS Transmitted Energy = -0.0 (dB)
OTFS Received Energy = -117.6 (dB)

PN Transmitted Energy = -0.0 (dB)
PN Received Energy = -119.0 (dB)
```

This matches results from earlier, albeit using much smaller delay and Doppler along with a 2 second symbol duration. These make waveform design more complicated yet the new enhancement to OTFS still improves energy transfer efficiency +1.4 dB versus a PN waveform that is not designed for the channel.

(*Note: this enhanced OTFS waveform is certainly not optimized. For this particular channel, one could change the frequency of each pulse to boost the efficiency further. One could alternatively design an LFM that would boost the energy efficiency. However, such approaches do not extend well to three or more paths in general, so these are not presented here.)*

Imperfect Phase Knowledge in Two-Path Delay-Doppler Channel

The enhanced OTFS waveform takes advantage of the delay-Doppler channel through superposition of fields that were emitted at different times into one time-of-arrival at the receiver. This boosts received energy. It also makes the approach sensitive to phase errors in the transmitter's estimate of the channel. Figure 6 compares the new OTFS enhanced waveform to a genetic PN waveform as the unbiased random phase error on each path estimate increases. The results are for the two-path channel with the second path shifted 0.1 milliseconds in delay and 2 Hz in Doppler with respect to the first path. With no phase error, the new OTFS extension transfers energy from transmitter to receiver +1.4 dB more efficiently than generic PN. The boost gradually degrades as the phase error per path increases. At worse than 80-degrees standard deviation, the new OTFS extension delivers the same energy as a generic PN signal.

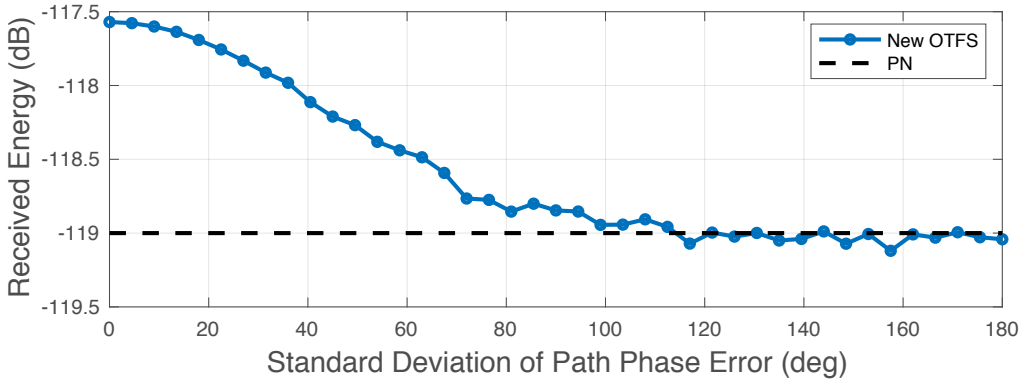


FIGURE 6: New OTFS benefits from accurate phase delivering energy up to +1.4 dB more efficiently vs. PN.

The following code produced Figure 6:

```
Fs = 1E6;                                   %sampling rate (samples/s)
Tsym = 2;                                   %transmission duration (s)
phaseStd = linspace(0, pi, 41);
TRIALS = 1000;
for test=1:length(phaseStd)
    test
  parfor trial=1:TRIALS
    tx = makeOTFS(tau, fd, a, Fs, Tsym); %build enhanced OTFS waveform
    tx = fliplr(conj(tx));               %time-reversed & freq. reversed
    aMod = a .* [exp(1i*phaseStd(test) *randn(1)), ... %phase error
                 exp(1i*phaseStd(test) *randn(1))];
    rx = channelModel(tau, fd, aMod, Fs, tx);          %pass thru channel
    E(trial,:) = reportEnergy(tx, rx, 'OTFS');         %report results
  end
  E = mean(10.^(E/10));                     %mean in (linear) units
  storeMeanTX_dB(test) = 10*log10(E(1));  %convert to (dB)
  storeMeanRX_dB(test) = 10*log10(E(2));  %convert to (dB)
end

figure;
subplot(211);
plot(rad2deg(phaseStd), storeMeanRX_dB);
```

Summary

For multi-path delay-Doppler channels, the efficiency of energy transferred from the transmitter to the receiver is a function of waveform design. A new extension of OTFS demonstrated one avenue to boost this energy efficiency as compared to a generic root-raised-cosine filtered PN waveform. For two paths with coefficients 1E-6 and 0.5E-6, the extension boosted energy transfer by +1.4 dB versus the generic signal. This performance advantage begins to erode when there is more than 20 degrees of error on the transmitter's knowledge of the path phases. For 80 degrees or more of such error, the designed waveform delivers about the same energy to the receiver as a generic waveform. Application gains from this approach depend upon channel estimation performance. As a result, the EME channel is particularly challenging due to 2.4 seconds of round-trip-delay and it requires further research into channel estimation approaches that are a separate problem from waveform design.

Acknowledgements

Thanks to Michelle Thompson for enlightening discussions and for hosting Open Research Institute meetings that inspire new ideas and potential advancements to the radio art. Thanks to Thomas Telkamp for sharing his Dwingeloo insights and data that inspired new ideas about delay-Doppler channels for amateur radio experimentation.

About the Author

For more than 20 years, Peter S. Wyckoff has designed and tested a wide variety of digital communications systems, modems, and antenna arrays, particularly for the satellite communications industry. He graduated from Penn State University with an MSEE in 2000 and graduated from Pitt with a BSEE in 1997. Since graduation, he has been awarded seven U.S. patents, which are mostly about co-channel interference mitigation, antenna array signal processing, and digital communications. In May 2023, his textbook "Visualizing Signal Processing with Complex Values" earned Amazon's #1 best-selling signal processing new release.

References

[1] Efficiently Using Transmit Symbol Energy via Delay-Doppler Channels — Part I, Open Research Institute Inner Circle Newsletter, May 2025, p. 12-13.
[2] Efficiently Using Transmit Symbol Energy via Delay-Doppler Channels — Part I, Open Research Institute Inner Circle Newsletter, May 2025, p. 15-16.
[3] A. Monk, R. Hadani, Mi. Tsatsanis, S. Rakib, OTFS — Orthogonal Time Frequency Space, https://arxiv.org/pdf/1608.02993
[4]. Efficiently Using Transmit Symbol Energy via Delay-Doppler Channels — Part I, Open Research Institute Inner Circle Newsletter, May 2025, p. 16.

Wireshark Plugin for Opulent Voice

See source code at:
https://github.com/MustBeArt/opv-wireshark-plugin/

An Opulent Voice station can be divided into a modem and a host that handles the protocol from audio down to the bytes in the 40ms Opulent Voice frames. Sometimes these two components are connected by Ethernet (or a more extensive network) and the bytes composing each frame are encapsulated in a UDP packet. This dissector helps Wireshark make sense of these encapsulated frames.

Each frame consists of two major parts: a frame header, and a payload. The frame header is a fixed format of 12 bytes. The payload contains bytes from a COBS-encoded stream of IP packets. In the common case where the payload contains Opus voice packets, and the voice packets are synchronized with the frame boundaries, the payload in a frame corresponds exactly to a single COBS-encoded IP/UDP/RTP/Opus packet. For other payload types, this is not true (except by coincidence). A frame's payload may consist of, in order, the last part of one packet, one or more whole short packets, and the first part of another packet. Any of these components may be omitted. The frame boundaries are delimited by a zero byte, according to the COBS protocol.

Because of the stream nature of the COBS-encoded packets, a complete dissector for this protocol will sometimes have to reassemble payload packets from the contents of multiple frames. Because the two ends of this stream connection are assumed to be part of the same receiving station, we can safely assume that the packets will arrive in order and intact a sufficiently high percentage of the time. If you are implementing a station design in which the modem and the host are connected over a larger, more complex network, it is your responsibility to take steps to ensure that these assumptions remain true. For instance, you might wrap this whole protocol in an outer COBS stream and send that stream via TCP.

Guide to Transmitting DVB-S2 Video using ORI Encoder

Please visit Aaron Olivarez's site below to read his report about using the ORI DVB-S2 encoder.

https://olivarez.info/blog/guide-to-transmitting-dvb-s2-video-using-ori-encoder/

Postlocutor, a Prototype Receiver for Opulent Voice

This is a simple receiver for the UDP-transported version of Opulent Voice frames created by Interlocutor, located at https://github.com/OpenResearchInstitute/interlocutor

Summary of Classes in the Program:

class OpulentVoiceProtocol (aka OPV)

OpulentVoiceProtocol encapsulates the Opulent Voice frame header and protocol knowledge. It provides a number of constant values and one method, parse_frame(), which extracts the fields of the frame header.

class AudioPlayer

AudioPlayer uses pyaudio (which uses PortAudio) to decode Opus voice packets and send the decoded audio samples to the default audio output device.

It manages a short queue of pending decoded audio frames. The queue is filled by calls to the method decode_and_queue_audio(), and emptied by callbacks to the method audio_callback() sent by itself.

It provides start() and stop() methods, which are called by the methods of the same names of the OpulentVoiceReceiver object.

It keeps some statistics, which can be copied out by calling the get_stats() method.

class OpulentVoiceReceiver

OpulentVoiceReceiver operates the receiver overall. It accepts the incoming UDP-encapsulated packets by opening a socket and operating a separate thread running the listen_loop() method repeatedly. That thread blocks on a recvfrom() call until a packet arrives, then calls process_frame() to handle it. process_frame() is also part of OpulentVoiceReceiver. This method should (but does not yet) feed the non-header portion of the UDP-encapsulated payload to the COBS decoder. If this results in the completion of a COBS packet, the COBS-decoded data from that packet is further analyzed by process_COBS_packet(), which checks for a valid UDP header and examines the destination IP address and port number. If this matches criteria for one of the services known to Opulent Voice, the packet is further processed according to the specified service. Otherwise, the packet is passed through to the host's network stack (but not yet).

When process_COBS_packet() handles a voice frame, it invokes AudioPlayer's decode_and_queue_audio() method to play back the received audio.

When it handles a text message, it fetches the station ID from the frame header, decodes it, and prints a line consisting of the station ID, a special icon marking this as a text message, and the text data from the packet payload.

When it handles a control message, it should (but does not yet) act on the contents of the control message. Currently, a control message is handled much like a text message.

Summary of Data/Control Flow:

1) main creates an OpulentVoiceReceiver. OpulentVoiceReceiver.__init__() instantiates an OpulentVoiceProtocol. OpulentVoiceReceiver.__init__() instantiates an AudioPlayer

2) main calls OpulentVoiceReceiver.start() and then goes into a loop calling time.sleep(1) until interrupted.

3) OpulentVoiceReceiver.start then creates and starts a daemon thread running OpulentVoiceReceiver.listen_loop() to receive the encapsulated Opulent Voice Packets.

4) listen_loop() sits in a loop calling recvfrom() on the UDP socket, which blocks and returns exactly one packet. This loop continues until self.running is false. Each time a packet is received, it is passed to self.process_frame()

5) process_frame() calls self.protocol.parse_frame() to extract the fields of the frame header (plus a timestamp) into a dictionary, which is returned.

The payload, which is an IP packet, is passed into Scapy, resulting in a Scapy packet called pkt. We check that it's an IP/UDP packet, confirm its checksums, and get the destination UDP port number. The port number is used to determine which service (voice, text, control) owns the packet.

If it's voice, we send the data to self.audio_player.decode_and_queue_audio(), which calls the Opus decoder in self.decoder.decode(), and adds the resulting frame of audio samples to self.audio_queue. Meanwhile, self.audio_player (which is an AudioPlayer) is pulling decoded audio frames from self.audio_queue via audio_callback(), and streaming them to the default audio output device.

If it's a text message, we decode the station_id to ASCII and print it with the text message data to the screen.

If it's a control message, we (currently) just output the text of the control message to the screen, along with the encapsulating host's IP address.

If, on the other hand, the packet isn't IP/UDP or its destination port isn't one known to Opulent Voice, we currently discard it. Eventually, this packet will be passed through to the host's network stack.

YouTube

https://www.youtube.com/@OpenResearchInstituteInc


RFBitBanger
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
HF Digital QRP
Get your kit today
while supplies last!
https://www.ebay.com/usr/openresearchinstitute
$175
Shipping
Calculated
Separately

The Inner Circle
Sphere of Activity

If you know of an event that would welcome ORI, please let your favorite board member know at our hello at openresearch dot institute email address.

30 June 2025 - Future Amateur Geostationary Payload Definitions comment deadline. Our submission was submitted on time and has been acknowledged.

20 Jul 2025 - Submission deadline for Open Source Cubesat Workshop, to be held 25–26 October 2025. Location is Serafio of the Municipality of Athens, Greece.

5 August 2025 - Final Technological Advisory Council meeting at the US Federal Communications Commission (FCC) in Washington, DC. The current charter concludes 5 September 2025.

7-10 August 2025 - DEFCON 33 in Las Vegas, Nevada, USA. ORI plans an Open Source Digital Radio exhibit in RF Village, which is hosted by Radio Frequency Hackers Sanctuary.

1 September 2025 Our Complex Modulation Math article will be published in ARRL's QEX magazine in the September/October issue.

5 September 2025 - Charter for the current Technological Advisory Council of the US Federal Communications Commission concludes.

19-21 September 2025 - ESA and AMSAT-DL workshop in Bochum, Germany.

25-26 October 2025 - Open Source Cubesat Workshop, Athens, Greece.

Thank you to all who support our work! We certainly couldn't do it without you.

Anshul Makkar, Director ORI
Frank Brickle, Director ORI (SK)
Keith Wheeler, Secretary ORI
Steve Conklin, CFO ORI
Michelle Thompson, CEO ORI
Matthew Wishek, Director ORI