SNR Calculation

Documentation for explaining UAVRT-SDR's current SNR calculations

Flagstaff, AZ
May 21, 2017

Edited by

Michael Finley
Northern Arizona University
Dynamic and Active Systems Laboratory

Contents

1 Project Description

The aim of this project is to construct an unmanned aerial system that integrates a radio telemetry receiver and data processing system for efficient detection and localization of tiny wildlife radio telemetry tags. Current methods of locating and tracking small tagged animals are hampered by the inaccessibility of their habitats. The high costs, risk to human safety, and small sample sizes resulting from current radio telemetry methods limit our understanding of the movement and behaviors of many species. UAV-based technologies promise to revolutionize a range of ecological field study paradigms due to the ability of a sensing platform to fly in close proximity to rough terrain at very low cost. The new UAV-based (UAV-RT) system will dramatically improve wildlife tracking capability while using inexpensive, commercially available radio tags. The system design will reflect the unique needs of wildlife tracking applications, including easy assembly and repair in the field and reduction of fire risk.

Our effort will focus on the development, analysis, real-time implementation, and test of software-defined signal processing algorithms for RF signal detection and localization using unmanned aerial platforms. It combines (1) aspects of both wireless communication and radar signal processing, as well as modern statistical methods for model+data inference of RF source locations, and (2) fusion-based inference that draws on RF data from both mobile UAV-based receivers and ground-based systems operated by expert humans.

The complete design, including parts, assembly plans, software, and training modules for the UAV-RT system will be open sourced for widespread adoption of the technology, with dissemination and outreach by a user community website and demonstrations at wildlife conferences.

1.1 Component Descriptions and Definitions

Airspy R2 Software Defined Radio (SDR) receiver, by Itead Studio (see: http://airspy.com/airspy-r2/)

Beacon Radio telemetry tag, provided by HOLOHIL (see: http://www.holohil.com/transmitters/tablebats)

LNA Low noise amplifier, provided by LNA4ALL

1.2 Document description

This document provides explanations for SNR calculations with the use of a software defined radio (SDR) receiver. This document will use the Airspy R2 as an example SDR receiver, but any SDR receiver should suffice assuming the correct drivers and dependencies can be acquired for usage with GNURadio. This document will also assume the use of GNURadio as a means of getting the actual I/Q data but many other SDR applications would work in its place (e.g. SDRSharp, etc.). This document will also be using MATLAB to calculate the SNR but open source programs like Octave will work as well. This document assumes the user has a raw binary I/Q data file readily available (see: I/Q Data Acquisition Guide) for processing.

1.2.1 Software

GNURadio GNURadio is a free and open source toolkit for software defined radio

GNURadio Companion GNU Radio Companion is a graphical user interface for GNURadio

MATLAB MATLAB, short for matrix laboratory, is a numerical computing environment

Octave Octave is a numerical copmuting environment very similar to MATLAB but open source

1.2.2 Hardware

Airspy R2 The Airspy R2 is an SDR receiver, specifications below

- 24-1800 MHz native RX range, down to DC with the SpyVerter option.
- 3.5 dB noise floor between 42 and 1002 MHz
- 10MSPS I/Q output, 16bit fixed or 32bit float ouput streams
- 2.5MSPS (experimental) I/Q output for low power devices
- 10MHz panoramic spectrum
- No I/Q imbalance, DC offset or 1/F noise
- SMA antenna connector
- Micro-B USB socket
- MCX connector for an external clock

2 SNR Explanation

2.1 Introduction

The requirement for an accurate signal-to-noise ratio (SNR) estimation technique is necessitated by the fact that various algorithms used in radio telemetry systems require knowledge of the SNR (see, for example, [1]-[3]) for optimal performance if the SNR is not constant. With knowledge of the SNR the performance of a system may be improved. SNR affects detection, tracking, and overall performance.

2.2 System Model

The objective is to create a method that estimates the SNR of a signal based on averaging observable properties of the received signal, below is a figure of the simplified block diagram of a general receiver (see: Figure 1). Radar signals are usually narrowband, bandpass, phase- or frequency-modulated functions [4].

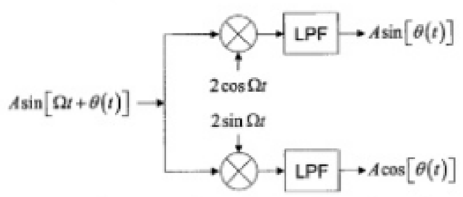

Figure 1: Simplified functional block diagram of receiver.

This implies the general formula for an echo waveform $r(t)$ received is of the form

$$r(t) = Asin[\Omega t + \theta(t)] \tag{1}$$

For our system more specifically the signals of interest are pulses of the form

$$p_k(t) = A_k^i sin[\omega_{RF} t + \theta_k^i(t)] * q_T^i(t - kT_{pri} - T_i) \tag{2}$$

Where A_k^i represents the amplitude modulation of the ith tag of the kth pulse, ω_{RF} represents the frequency, θ_k^i represents the phase modulation for the ith tag and kth pulse, t represents time in the continuous-time domain, T_{pri} represents the pulse repition interval, and T_i represents the pulse duration. The piece-wise definition of the pulse function q_T^i is:

$$q_T^i = \begin{cases} 1 & 0 \leq t \leq 1 \\ 0 & else \end{cases}$$

The sum of all individual tags, index i, and the sum of all individual pulses, k, for each tag can be summed:

$$x_{RF} = \sum_k \sum_i p_k^i(t) + n(t) \tag{3}$$

For the sake of developing the trigonometric simplification used to represent our received signal we will assume only one beacon is being received on the antenna, and that the signal is constant, not a pulse, simplifying equation (2) to:

$$p_k(t) = A_k^i sin[\omega_{RF} t + \theta(t)] \tag{4}$$

We assume the Airspy R2 can be modeled as a conventional quadrature channel receiver model. In Figure 1, above, the lower channel is the in-phase ("I") channel, and the upper is the quadrature phase ("Q") channel. The received signal is split into two channels. One channel, called the *in-phase* or "I" channel of the receiver, which mixes the received signal with an oscillator, called the *local oscillator* at the radar frequency, this generates both sum and difference frequency components based on a trigonometric identity.

$$2sin[\omega_{RF} t] * A_k(t) sin[\omega_{RF} t + \theta(t)] = A_k(t) cos[\theta(t)] - A_k(t) cos[2\omega_{RF} t + \theta(t)] \tag{5}$$

The sum term is removed by the lowpass filter, leaving only the modulation term $Acos[\Omega t]$. The second channel, called the *quadrature* phase or "Q" channel, mixes the signal with an oscillator having the same frequency but with a 90°phase shift from the I channel oscillator. The Q channel mixer output is

$$2cos[\omega(t)] * A_k(t) sin[\omega_{RF} t + \theta(t)] = A(t) sin[\theta(t)] - A_k(t) sin[2\omega_{RF} t + \theta(t)] \tag{6}$$

The sum term is, again, removed by the lowpass filter, leaving the modulation term $A(t)sin[\Omega t]$. The general convention in modeling the input signal $r(t)$ is that the I channel is the one where the oscillator function (sine or cosine) is the same as that used in modeling the signal, hence the name in-phase. That is to say, if you modeled your input signal as $Acos[\Omega t + \theta(t)]$ the I and Q channels would be flipped.

The justification for both I and Q channels is that either one alone does not provide enough information to determine the phase modulation term $\theta(t)$ unambiguously. When both I and Q channels are implemented, the phasor quadrant is determined unambiguously. Signal processors normally assign the I signal to be the real part of a complex signal and the Q signal to be the imaginary part, forming a single complex signal

$$x_{RF}(t) = I(t) + jQ(t) = e^{j\theta(t)} \tag{7}$$

Equation (7) shows a convenient representation of an ideal coherent receiver on a transmitted signal. Several assumptions are made in this model. One being that the I and Q channels have perfectly matched transfer functions over the signal bandwidth. This implies that the gain through each of the two signal paths must be identical, as must be the phase delay of the two channels. Real receivers do not have perfectly matched channels. For the purposes of calculating the SNR of this system we can assume the negative effects are negligible. Another requirement is that the oscillators used to demodulate the I and Q channels must be exactly in quadrature, 90°out of phase with one another.

As shown in equation (7), $x_{RF}(t)$ is a complex representation of the continuous time signal. In order to compute the SNR of the system one must obtain digital readings by an analog to digital converter (ADC) for a representation of the discrete-time signal, $x[n]$. The ADC conversions of our signal are digital representations of voltages on the receiving antenna of our radio front end. The complex representation of $x[n]$ is below.

$$x[n] = I(t) + jQ \tag{8}$$

2.3 Signal to Noise Ratio

In order to calculate the SNR of the signal one must take into consideration the magnitude of the complex discrete-time signal, which can also be understood as the envelope of the signal.

$$|x[n]| = x^{'}[n] = \sqrt{I(t)^2 + jQ(t)^2} \tag{9}$$

We can interpret our system's signal as a discrete-time signal $x[n]$ consisting of the sum of a "desired" signal, $s[n]$, and a "noise" signal, $w[n]$:

$$x^{'}[n] = s[n] + w[n] \tag{10}$$

The SNR of the signal can be defined as the ratio of the power of the desired signal, $s[n]$, to that of the noise signal, $w[n]$. To obtain the power of the desired, S, and the power of the noise signal, N, we calculate the powers using standard formulas:

$$S = \frac{x'[n]^2}{R} = \frac{s[n]^2}{R} \tag{11}$$

$$N = \frac{x'[n]^2}{R} = \frac{w[n]^2}{R} \tag{12}$$

Where R is some constant resistance of the antenna. The ratio of the power of the desired signal over the ratio of the power of the noise signal:

$$SNR = \frac{\frac{s[n]^2}{R}}{\frac{w[n]^2}{R}} \tag{13}$$

Which reduces to:

$$SNR = \frac{s[n]^2}{w[n]^2} \tag{14}$$

With the formal SNR equation defined, SNR can be written symbolically as shown below:

$$SNR = \frac{S}{N} \tag{15}$$

For convenience SNR is measured on a logarithmic scale, in decibels, augmenting equation (7) to be:

$$SNR = 10log_{10}(\frac{S}{N}) \tag{16}$$

To determine S and N for our system we will take into consideration the form of our received signal: a pulse with a specific pulse duration, T_i, and a specific pulse repetition interval, T_{pri}. We interpret our signal to be $x[n]$ then assume what specific sample indices, n, that are $s[n] + w[n]$ (signal and noise) or are solely $w[n]$ (just noise) through visual inspection based on our knowledge of the signal being received from the beacons in the system. The overall structure of a discrete signal can then be assumed knowing the pulse width and time delay between pulses.

The noise average power is needed first to subtract from the signal plus noise average power. The noise average power is represented as

$$N_{sum} = \frac{1}{(T_{pri} - T_i) * (n_p + 1)} \left(\sum_{n=1}^{n_{p(1)}} x'[n] + \sum_{n_{p(1)}+T_i}^{n_{p(1)}+T_{pri}} x'[n] + \sum_{n_{p(1)}+T_i+T_{pri}}^{n_{p(1)}+2T_{pri}} x'[n] + ... + \sum_{n_{p(1)}+T_i+2T_{pri}}^{n_{p(1)}+(j-1)T_{pri}} x'[n] \right) \tag{17}$$

Where n is defined as a specific sample index for $1 \leq n \leq j$ where j represents the last index of the discrete time sampled signal, T_{PRI} is the number of samples in a single pulse width from $x[n]$ to $x[n + T_{PRI}]$, T_{PRI} is the pulse delay between pulses starting, n_p represents the number of pulses in the discrete time signal, n_{p1} represents the specific sample index that the first pulse starts at. With these variables determined the formula of a discrete signal's SNR can then be constructed. The sum is divided by $((n_p + 1) * (T_{pri} - T_i))$ because for a discrete time signal with n_p pulses there would be $n_p + 1$ sections of noise of duration $(T_{pri} - T_i)$.

The signal power is represented as

$$S = \frac{1}{T_i * n_p}\left(\sum_{n=n_{p(1)}}^{n_{p(1)}+T_i} x'[n] + \sum_{n_{p(1)}+d}^{n_{p(1)}+T_i+T_{pri}} x'[n] + \sum_{n_{p(1)}+2T_{pri}}^{n_{p(1)}+T_i+2T_{pri}} x'[n] + ... + \sum_{n_{p(1)}+(j-1)T_{pri}}^{n_{p(1)}+T_i+(j-1)T_{pri}} x'[n]\right) - N \quad (18)$$

Where all variables used in the representation of the noise average power, equation (16), are the same for equation (17). The sum is divided by $((n_p) * (T_i)$ because for a discrete time signal with n_p pulses there would be n_p sections of noise of duration T_i.

Variables T_i and T_{pri} have units of 'samples', and can be discovered by knowing the sampling rate of the systems ADC, fs, and the time domain pulse width interval, t_i, and the time domain pulse repetition interval, t_{pri}.

$$T_i = fs * t_i \quad (19)$$

$$T_{pri} = fs * t_{pri} \quad (20)$$

For example, in our system, a pulse interval in the time domain of $.18mS$ with an ADC sample rate of 10 million samples per second (Msps) would have a T_i of 1800 samples. A time domain pulse repetition interval of 36ms with an ADC sample rate of 10 million samples per second (Msps) would have a T_{pri} of 360,000 samples.

2.4 Conclusions

This approach to developing a model for SNR is relatively reductive in the sense that it strips SNR down to a fundamental perspective. For more intelligent approaches to calculating SNR see [5] for a discussion of several methods with varying levels of performance and accuracy as trade-offs. Some algorithms are even more suited for specific types of modulated signals.

Something worth noting in this approach is that current methods rely on the assumption that the first pulse will be distinguishable through visual inspection, as well as the number of pulses will be distinguishable through visual inspection, this isn't guaranteed for continuous time signals with a SNR significantly lower than 0. This is a common downside to many SNR estimators; accuracy tends to suffer at low levels of SNR.

2.5 References

[1]D. G. Brennan,"Linear diversity combining techniques," *Proc. IRE*, vol. 47, pp. 1075-1102, June 1959.

[2]J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc. IEEE Global Telecommunications Conf.*, Dallas, TX, Nov. 1989, pp. 1680-1686.

[3]P. A. Wintz and E. J. Luecke, "Performance of optimum and suboptimum synchorinzers," *IEEE Trans. Commun.*, vol. COM-17, pp. 380-389, June 1969.

[4]M. Richards "Fundamentals of Radar Signal Processing," pp. 15-19

[5]D. Pauluzzi and N. C. Beaulieu, "A Comparison of SNR Estimation Techniques for the AWGN Channel," *IEEE Xplore*, vol. , pp. 1681-1691, 2000.

3 Methods/Procedures

3.1 Installing Necessary Software

3.1.1 MATLAB

To install MATLAB reference mathwork's installation guides (see: https://www.mathworks.com/support/install-matlab.html). Installing MATLAB requires a valid software license, which can be obtained by purchasing the product from the MathWorks Store (see: https://www.mathworks.com/store/) or downloading a product trial.

3.1.2 Octave

If MATLAB is not available to the user, Octave is a free open source numerical computing environment with similar syntax to MATLAB. To install Octave on a Windows machine you will want to download the Windows binaries with corresponding source code (see: https://www.gnu.org/software/octave/download.html) and navigate to the following link (see: https://ftp.gnu.org/gnu/octave/windows/) and select the user's preferred installation method. The "octave-4.2.1-w64-installer.exe" is recommended (as of 3/5/2017, this is Octave's newest stable version).

All software installations are now complete for calculating SNR.

4 MATLAB Processing

After opening MATLAB navigate to the directory in which the I/Q data file was saved. In order for the user to use or begin processing the I/Q data for SNR. The function used for this is included below.

The following MATLAB command would save `my_data_SNR` to your workspace as a float variable containing the average SNR of the discrete time signal IQ file taken in by the function.

```
my_data_SNR = SNR_calculation('file_sink_tutorial', 1.02e7, 5, .018e7, 1.2945e7);
```

The desired function is detailed below for reference:

```

function SNR = calculate_SNR (filename1, fp, num_pulses, w, d)

  %% usage: SNR = calculate_SNR (filename1, fp, num_pulses, w, d)
  %%
  %% SNR, an integer variable, will be stored in your workspace for
  %% convenience of tracking SNR at different meaningful distances or
  %% circumstances surrounding the VHF beacon
  %%
  %% fp is the sample, or array index, of the first pulse
  %% num_pulses is the number of pulses seen, visually from
  %% w and d are the pulse width and time delay between pulses
  %% w and d should be an integer representing the number of samples for w and d
  %% plot(abs(IQ_file))
  %% after the IQ file has been read with
  %% IQ_file = read_complex_binary_gnu_radio('IQ_data_file')
  %%
  s_and_n = read_complex_binary (filename1); %reading IQ file

    function v = read_complex_binary (filename, count) %read IQ function
        m = nargchk (1,2,nargin);
        if (m)
        usage (m);
        end

        if (nargin < 2)
        count = Inf;
        end

        f = fopen (filename, 'rb'); %opening raw binary file
        if (f < 0) %checking for data
         v = 0;
        else
        t = fread (f, [2, count], 'float'); %reading as a float
        fclose (f); %closing file
        v = t(1,:) + t(2,:)*i; %assuming IQIQIQ structure of binary file
        [r, c] = size (v); %creating matrix to match IQ file size
        v = reshape (v, c, r); %reshaping matrix to correct format
        end

    end
n_samples = length(abs(signal_and_noise)); %number of samples in stream
p_noise = 0; %initializing variable p_noise (Noise Power)
p_signal = 0; %initializing variable p_signal (Signal Power)
for n = 0:1:num_pulses
switch n
    case 0 %formula needed when calculating p_noise for first section
        p_noise = p_noise + mean(abs(s_and_n(1:fp)));
        p_signal = p_signal + mean(abs(s_and_n(fp + (n * 1.295e7):fp + w + (n * d))));
    case (n > 0) && (n ~= num_pulses)
        p_noise = p_noise + mean(abs(s_and_n(fp + w + ((n-1) * d):fp + w + ((n)*d))));
        p_signal = p_signal + mean(abs(s_and_n(fp + (n * d):fp + w + (n * d))));
    case num_pulses
        p_noise = p_noise + mean(abs(s_and_n(fp + w + ((n-1) * d):n_samples)));
        p_signal = p_signal + mean(abs(s_and_n(fp + ((n-1) * d):fp + w + ((n-1)*d))));
end

end
p_signal = p_signal/num_pulses; %taking average power of signal
p_noise = p_noise/(num_pulses + 1); %taking average power of noise
p_signal = p_signal - p_noise; %subtracting avg noise power from avg signal power

%variable to be returned to variable calling function
SNR = 20 * log10(p_signal/p_noise); %SNR calculation, 20 comes from P = V^2/R

%code for plotting IQ file with marked pulses
```

```
plot(abs(signal_and_noise)) %plot IQ_file
title('Signal Inlab (w/o LNA)') %plot title
ylabel('Amplitude (V)') %y-axis label
xlabel('Samples @10Msps') %x-axis label
legend(['SNR = ' num2str(SNR) ' (dB)']) %legend to display SNR

hold on
for n = 0:1:num_pulses
    switch n
        case 0
            plot(first_pulse, .02, 'r-o');
            plot(first_pulse + (n * 1.295e7), .02, 'r-o');
        case num_pulses
            plot(first_pulse + .2e6 + ((n-1) * 1.295e7), .02, 'r-o');
            plot(first_pulse + ((n-1) * 1.295e7), .02, 'r-o');
        otherwise
            plot(first_pulse + .2e6 + ((n-1) * 1.295e7), .02, 'r-o');
            plot(first_pulse + (n * 1.295e7), .02, 'r-o');
    end

end
hold off
end
```

Essentially this function calls a nested function that opens the desired file, designates that is raw binary, reads the raw binary file as a float, closes the file, then parses the data assuming it is structured like I1Q1I2Q2I3Q3... until the end of the file is reached. Then the function reshapes it as desired. What is left is an output in complex form with I being the real part and Q being the imaginary part in its complex representation.

The function then brings to fruition equation (16) and equation (17) with the use of a case/switch structure. Essentially the summations that occur for signal and noise are identical save for one variable. The only exceptions to this are the first area of noise (from the first sample to the first pulse) and the last area of noise (from the end of the last pulse to the end of the discrete time signal. These two areas of noise are accounted for with case n = 0 of the iteration, and the case where the last pulse has been accounted for.

The above MATLAB code includes a section that plots the IQ data along with markers to signify where the program thought pulses were, and it also populates the graph with a legend detailing the SNR of the discrete time signal.

5 Recommendations

Inclusion of details on using the FFT MATLAB function to quantify frequency content. Right now I need to do some more research on how the FFT works so I can speak more confidently about what the FFT is showing. Adding a vertical line at the start and end of what are considered to be pulses would probably be more visually understandable as opposed to a single point with a predetermined y-value being plotted on top of the IQ data. With large IQ data files the markers would get buried fairly easily. Also some refactoring of code to determine a way to identify pulses without user interference or interpretation. That way the system could become completely automated.

Currently this program also assumes a constant sampling rate, this function can be augmented to take

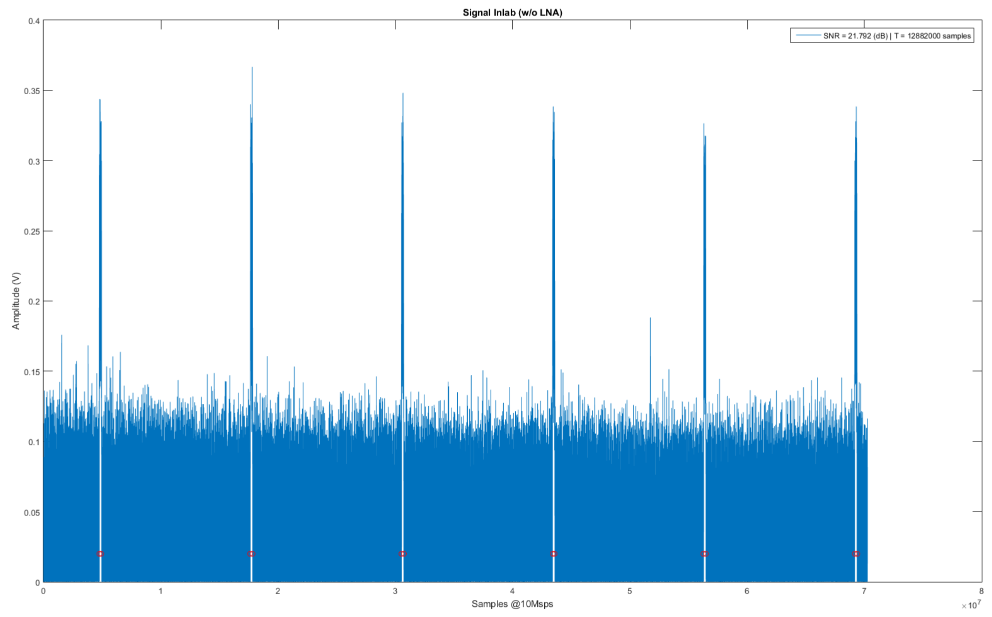

Figure 2: MATLAB output of IQ data with no LNA, SNR included in legend

in as parameters values for sampling rate, T_{pri} and T_i.