I/Q Data Guide v.1

Guide for acquiring I/Q data with the Gnu Radio Framework

Flagstaff, AZ
March 5, 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, by HOLOHIL (see: http://www.holohil.com/transmitters/tablebats)

LNA Low noise amplifier

1.2 Document description

This document provides instructions for someone to reproduce inphase quadrature (I/Q) data with the use of their own 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. 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. This document will also be using MATLAB to process the I/Q data but open source programs like Octave will work as well. This document's version also assumes the use of a Windows PC but will be added to with documentation on acquiring I/Q data with other platforms and SDR software (i.e. Linux with GNURadio, Linux with Gqrx, Raspbian with GNURadio, Raspbian with Gqrx, or Windows with SDRSharp).

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

SDRSharp SDRSharp is a simple DSP application for SDR.

Gqrx Gqrx is an open source SDR receiver powered by the GNU Radio toolkit and Qt graphical tookit

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 Methods/Procedures

2.1 Installing Necessary Software

2.1.1 GNURadio and GNURadio Companion

First install GNU Radio and GNU Radio Companion from gnuradio.org. GNU Radio's site has tutorials for installing GNU Radio (see: http://gnuradio.org/redmine/projects/gnuradio/wiki/InstallingGR). But for the sake of this tutorial a guide for installing GNURadio and GNURadio Companion is what follows. It's worth noting GNU Radio does not officially support windows. The current recommended approach to install GNURadio and GNURadio Companion is from the binary installers, which can be found here (see: http://www.gcndevelopment.com/gnuradio/downloads.htm), select the download from the "Windows Installer" row and the "64-Bit Any CPU" column. This installer includes all dependencies for Windows, a custom python distribution, commonly used SDR drivers, and several OutOfTree (OOT) modules which allow the use of signal processing "blocks" that don't live within the GNU Radio source tree (see: http://gnuradio.org/redmine/projects/gnuradio/wiki/OutOfTreeModules). One of these OOT modules is the osmocom source which will be detailed on later.

Once the binary installer has finished downloading, run the .msi file and a setup wizard for GNU Radio should start. Follow the instructions of the setup wizard.

2.1.2 Airspy R2 Source Blocks (osmocom source)

As noted before this tutorial uses the Airspy R2 as an example SDR receiver for acquiring I/Q data. Unfortunately GNU Radio requires supplemental processing blocks to use the Airspy R2 as a receiver. The supplemental source block, "osmocom source" was intended for the OsmoSDR receiver but works with many other SDR receivers if their drivers and dependencies are installed. The source block of interest is the "osmocom source" which has support for the Airspy R2's R820t tuner through libairspy (see: https://github.com/airspy/host). Fortunately the dependencies included in the Windows binary installers include the osmocom source block and it's necessary dependencies to use the Airspy R2.

2.1.3 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.

2.1.4 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 acquiring I/Q data.

2.2 GNURadio Companion

This guide will not assume a working knowledge of how the processing blocks in GNURadio work. GNURadio Companion is a convenient tool to visually place together GNURadio's processing blocks. GNURadio processing blocks are written in C++ and work together with Python's fast scripting language. This tutorial will utilize the visual aid of GNURadio Companion.

Starting GNURadio Companion should eventually open up a GUI, as shown in Figure 1. To acquire I/Q data add the "osmocom Source" block to your workspace, to find this block search the Library with "ctrl + f" and select the "osmocom Source" by double clicking it, this will add it to the workspace. To configure this block for the Airspy R2 specifically, we need to input Device Arguments. Double click the source block once it is in the workspace, under Device Arguments input "airspy=0,bias=0". What this does is let the source know you have an airspy, and that you want disable the DC bias at the antenna input. The DC bias can be enabled by altering the argument to be "bias=1". The user should input the desired frequency to tune the antenna with "Ch0: Frequency (Hz)". This input field accepts scientific notation (e.g. 150.632e6 for 150.632 MHz). Select "Apply" and then select "Ok" to close the window.

Next we need a file sink to save the I/Q data somewhere for processing later. Search in the library window for "File Sink" using "ctrl + f". Double click the "File Sink" option to add it to the workspace. To connect our source block to our sink block the user should press "ctrl" and select the blue output of the source block and then also select the blue input of the sink block. These colored output and input squares that are attached to the processing blocks represent inputs and outputs, different colors represent different data types. For this tutorial's example the blue input and output represent a complex float data type. To select a directory in which the raw binary file will be saved from the file sink, double click the "File Sink" block and select the "..." button, navigate to the desired file directory and input a file name under the "Name:" option.

Lastly we need to initialize a sample rate variable for our workspace. By default GNURadio Companion initializes a sample rate variable which we will edit to be the Airspy R2's sample rate as an example for this document. To do this double click the "Variable" block and set the "Value" parameter to "10e6" (i.e. 10MSPS). The user will notice by default our source block takes in this variable as it's sampling rate. This is particularly useful as a user could initialize multiple variables for the use of a GNURadio Companion file

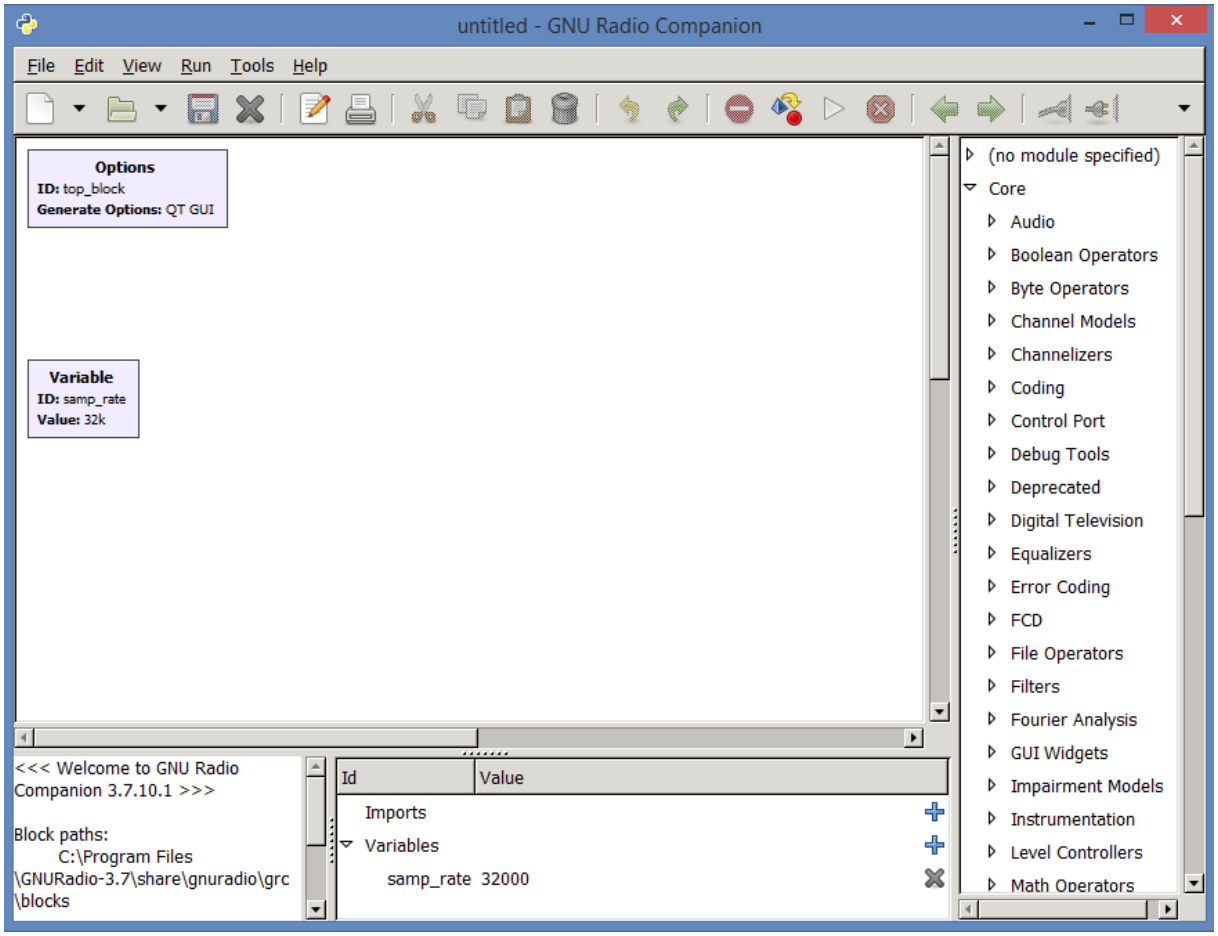

Figure 1: Default file start of GNURadio Companion

that utilizes multiple sampling rates (e.g. recording Audio input at 44.1kSPS while also recording I/Q data at 10MSPS). Save your GNURadio Companion file, see Figure 2 as a reference for what the general I/Q data file sink we've just created looks like in GNURadio Companion.

Now the user can execute their flow graph by pressing the play button in the tool bar of GNURadio Companion. Be sure that the Airspy R2 is plugged in through a USB cable connected to the Micro-B USB socket of the Airspy R2. Also ensure that the beacon of interest has power being supplied to it so it emits its RF signal. After the program has executed, which for this file must be manually done by pressing the "Kill the flowgraph" button (the stop sign with a white 'X' in it on the toolbar), the directory the user previously selected in the "File Sink" block is where the raw binary I/Q data is now saved.

3 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 the user must read the complex binary file generated by GNURadio. With regards to MATLAB and Octave, Octave is the most popular analysis tool with GNU Radio, as the GNU Radio package includes its own set of scripts for reading and parsing output. Luckily the script of interest for processing complex binary, written in Octave, works just as well in MATLAB.

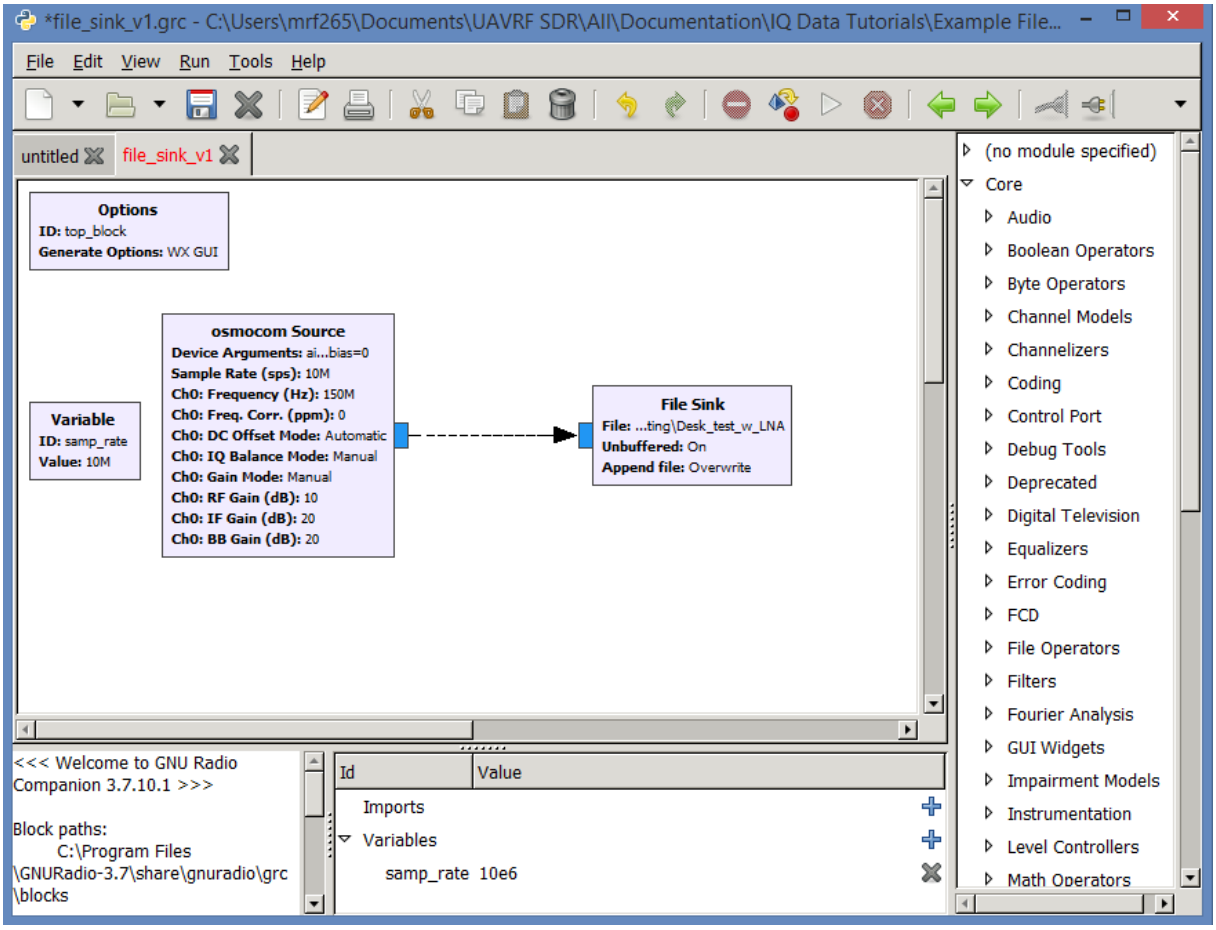

Figure 2: Generic I/Q data file sink .grc workspace

In order to use the script, or function in MATLAB, move the "read_complex_binary.m"file to the same directory the I/Q data file is located. To navigate to the "read_complex_binary.m"file go to "C:\...\Gnu\gnuradio-3.7.5\gr-utils\octave" and copy the desired file to the I/Q data file directory. The ... in the previous path designates whereever the user selected to install GNURadio. Your "Current Folder" in MATLAB should now look something like Figure 4 but with the user's selected I/Q data file name. Now with the desired function in our Current Folder we can begin processing the user's I/Q data with the following MATLAB command:

```
my_data_complex = read_complex_binary('file_sink_tutorial');
```

my_data_complex is the processed I/Q data, where file_sink_tutorial is the user selected file name for the I/Q data. A new variable with the name my_data_complex should appear in your workspace as a 'number of samples'x1 complex double. The number of samples is determined by the users SDR receiver and sampling rate used. The desired function is detailed below for posterity:

```
function v = read_complex_binary (filename, count)

  %% usage: read_complex_binary (filename, [count])
  %%
  %%  open filename and return the contents as a column vector,
  %%  treating them as 32 bit complex numbers
  %%
```

```
  m = nargchk (1,2,nargin);
  if (m)
    usage (m);
  end

  if (nargin < 2)
    count = Inf;
  end

  f = fopen (filename, 'rb');
  if (f < 0)
    v = 0;
  else
    t = fread (f, [2, count], 'float');
    fclose (f);
    v = t(1,:) + t(2,:)*i;
    [r, c] = size (v);
    v = reshape (v, c, r);
  end
```

Essentially this function 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 user may now start viewing an manipulating the I/Q data. To view the I and Q data separately the user may execute the following in MATLAB:

```
subplot(2,1,1)
plot(real(ans))
title('I Data')
xlabel('Samples @10MSPS')
ylabel('Amplitude')
subplot(2,1,2)
plot(imag(ans))
title('Q Data')
xlabel('Samples @10MSPS')
ylabel('Amplitude')
```

MATLAB's output should be something similar to Figure 3.

The user can zoom in on a specific pulse to view the frequency content by inspection as well and the results should be something similar to Figure 4, depending on the RF signal from the beacon and the frequency selected for the antenna.

4 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.

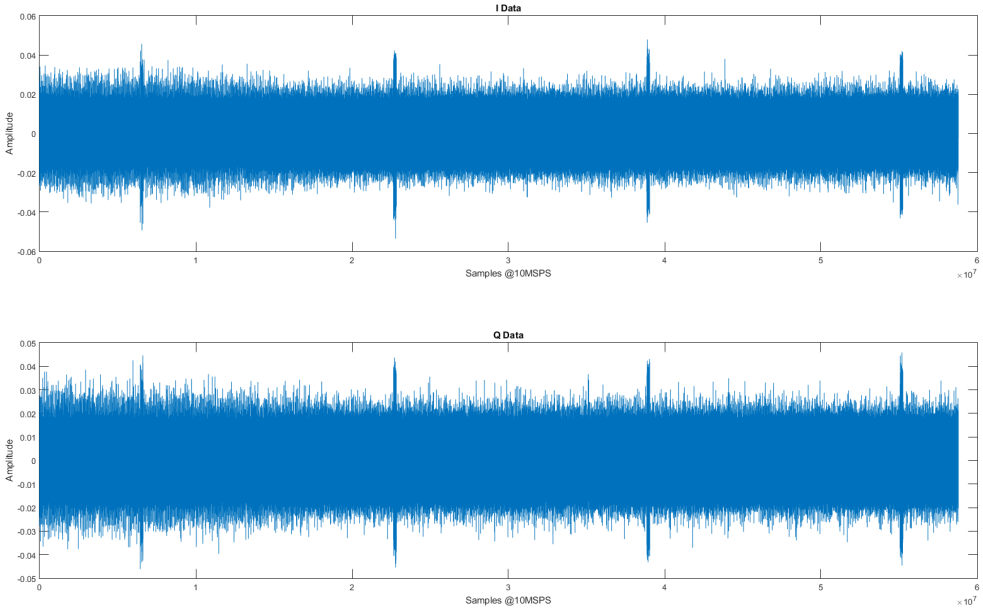

Figure 3: MATLAB output of I and Q data on separate plots

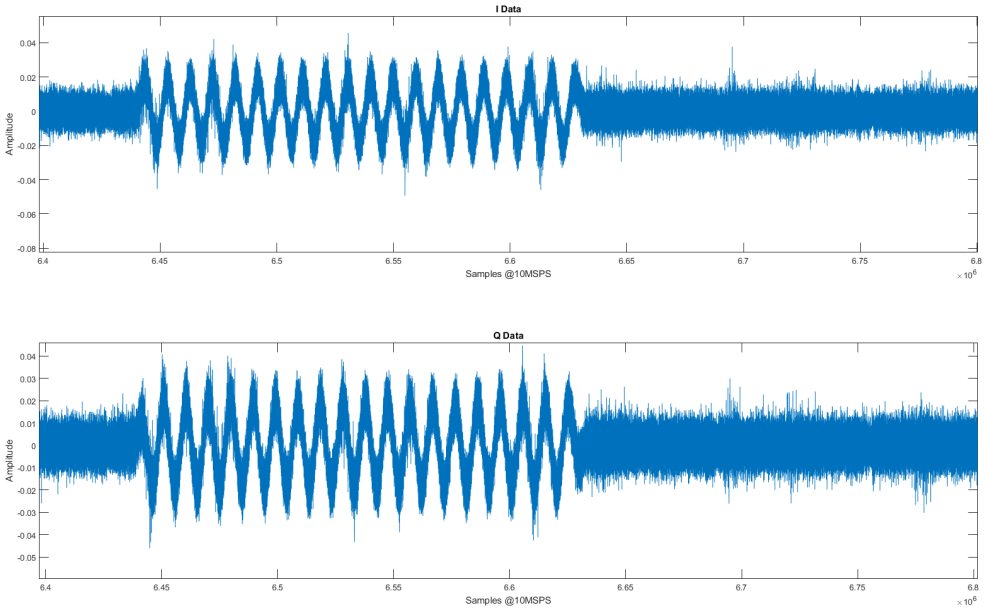

Figure 4: MATLAB output of I and Q data on separate plots, zoomed in on a pulse to view frequency