Peidong Qi
05/08/2018
AMRUPT, Spring 2018

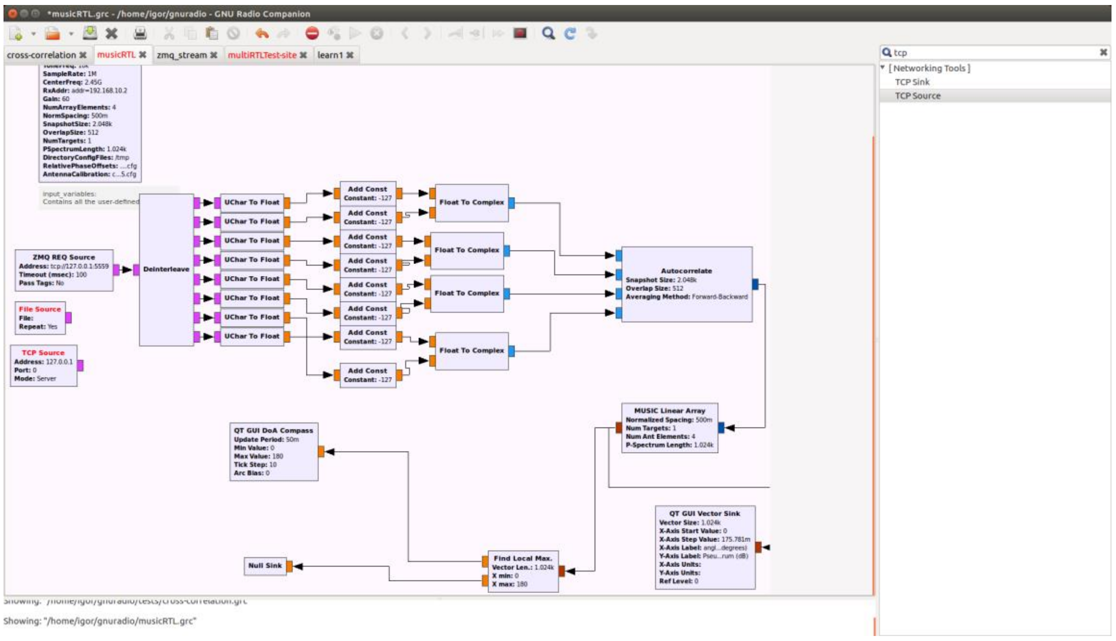
RTL SDR to Raspberry Pi Connection and Datalogging

Goals

Run Ettus's GRC project on Raspberry pi. Get the phase difference of two signals.

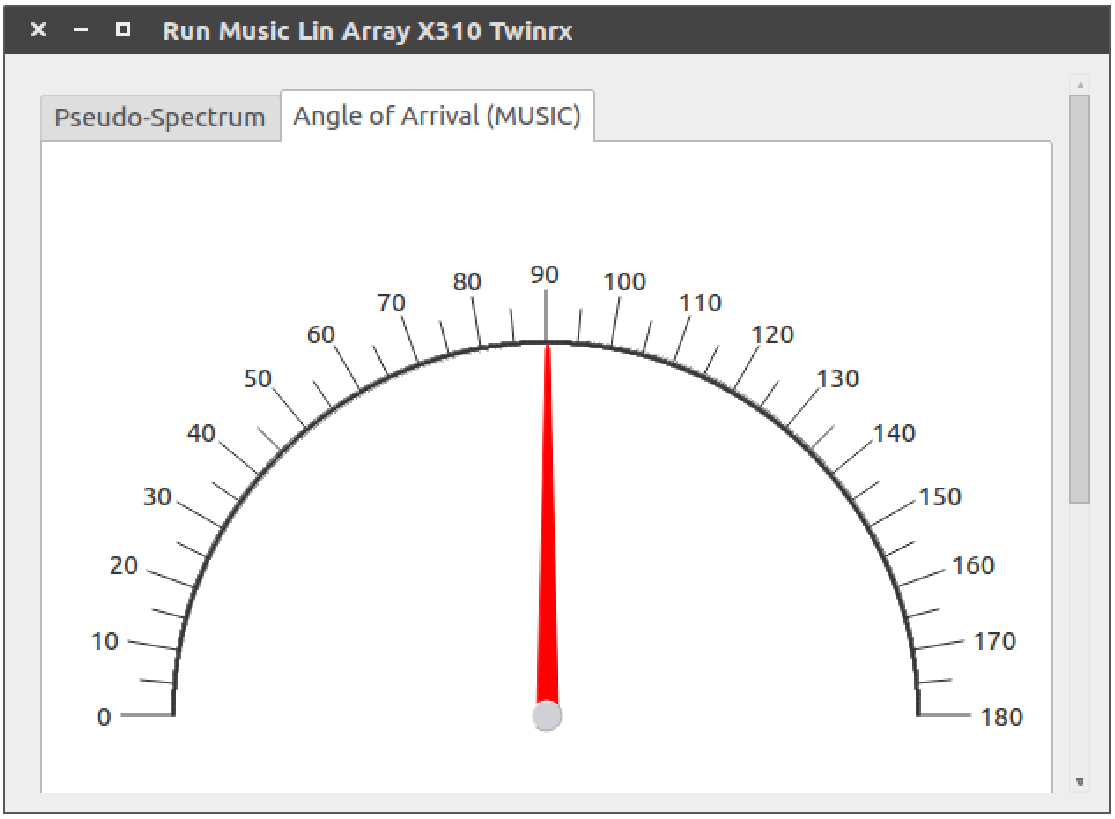
General approach

For the last week, we have tried on run Ettus's direction-finding project. We have tried serval different platform to run GRC. So far, we have tried Raspbian, Ubuntu MATE, Ubuntu Core, Windows. Unfortunately, we have run different problems with those operating systems. For Raspbian, we have problem about the miss library which the Ubuntu have. After we checked Ettus's configuration, we found out the Ettus run their project on Ubuntu operating system. Then we switched to ubuntu system. At beginning, we installed Ubuntu MATE on Raspberry Pi3. Then we installed everything we needed to run Ettus's project. However, we have another problem that we can't import error for the Ubuntu operating system. So we decided to switched to Ubuntu Core. After we installed everything, we found out that Ubuntu Core didn't support GUI interface. And the GRC need GUI interface to run. We have tried a lot of solutions to install GUI interface on Ubuntu Core. However, none of those solutions works. Then we installed Ubuntu on the PC, the import error has fixed. But the PC didn't recognize the SDR cards since it is the virtual system. Then we use VM to be our final chose. Russell has successfully run the GRC on VM.



```
SampleRate: 1M
CenterFreq: 2.45G
RxAddr: addr=192.168.10.2
Gain: 60
NumArrayElements: 4
NormSpacing: 500m
SnapshotSize: 2.048k
OverlapSize: 512
NumTargets: 1
PSpectrumLength: 1.024k
DirectoryConfigFiles: /tmp
RelativePhaseOffsets: ....cfg
AntennaCalibration: c...5.cfg

ZMQ REQ Source
Address: tcp://127.0.0.1:5559
Timeout (msec): 100
Pass Tags: No

File Source
File:
Repeat: Yes

TCP Source
Address: 127.0.0.1
Port: 0
Mode: Server

Add Const
Constant: -127

Autocorrelate
Snapshot Size: 2.048k
Overlap Size: 512
Averaging Method: Forward-Backward

MUSIC Linear Array
Normalized Spacing: 500m
Num Targets: 1
Num Ant Elements: 4
P-Spectrum Length: 1.024k

QT GUI DoA Compass
Update Period: 50m
Min Value: 0
Max Value: 180
Tick Step: 10
Arc Bias: 0

QT GUI Vector Sink
Vector Size: 1.024k
X-Axis Start Value: 0
X-Axis Step Value: 175.781m
X-Axis Label: angl...degrees)
Y-Axis Label: Pseu...rum (dB)
X-Axis Units:
Y-Axis Units:
Ref Level: 0

Find Local Max.
Vector Len.: 1.024k
X min: 0
X max: 180

Showing: "/home/igor/gnuradio/tests/cross-correlation.grc"
Showing: "/home/igor/gnuradio/musicRTL.grc"
```

The picture above shows how the Ettus's project works. For our own project, we don't need ZMQ REQ Source block to transfer data between SDR cards and host since we don't need wireless data transfer. Due to this week, we will get the direction-finding project works. We use CC1310 to be transmitter to send out the 433 MHZ frequency signal. And Adjust the position of CC1310, check if we can get the right angle.



Source code for key blocks

autocorrelate_impl.cc

```
#  ifdef HAVE_CONFIG_H
# include "config.h"
#  endif
# include < gnuradio / io_signature.h > #include "autocorrelate_impl.h"
#  include < armadillo > #define COPY_MEM false
// Do not copy matrices into separate memory
# define FIX_SIZE true
// Keep dimensions of matrices constant

namespace gr {
    namespace doa {

        autocorrelate::sptr
        autocorrelate::make(int inputs, int snapshot_size, int overlap_size, int av
g_method) {
            return gnuradio::get_initial_sptr(new autocorrelate_impl(inputs, snapsh
ot_size, overlap_size, avg_method));
```

```cpp
        }

        /*
         * The private constructor
         */
        autocorrelate_impl::autocorrelate_impl(int inputs, int snapshot_size, int overlap_size, int avg_method): gr::block("autocorrelate",
                gr::io_signature::make(inputs, inputs, sizeof(gr_complex)),
                gr::io_signature::make(1, 1, sizeof(gr_complex) * inputs * inputs)),
            d_num_inputs(inputs),
            d_snapshot_size(snapshot_size),
            d_overlap_size(overlap_size),
            d_avg_method(avg_method) {
                d_nonoverlap_size = d_snapshot_size - d_overlap_size;
                set_history(d_overlap_size + 1);

                // Create container for temporary matrix
                d_input_matrix = arma::cx_fmat(snapshot_size, inputs);

                // initialize the reflection matrix
                d_J.eye(d_num_inputs, d_num_inputs);
                d_J = fliplr(d_J);
            }

        /*
         * Our virtual destructor.
         */
        autocorrelate_impl::~autocorrelate_impl() {}

        void
        autocorrelate_impl::forecast(int noutput_items, gr_vector_int & ninput_items_required) {
            // Setup input output relationship
            for (int i = 0; i < ninput_items_required.size(); i++)
                ninput_items_required[i] = d_nonoverlap_size * noutput_items;
        }

        int
        autocorrelate_impl::general_work(int output_matrices,
            gr_vector_int & ninput_items,
            gr_vector_const_void_star & input_items,
            gr_vector_void_star & output_items) {
            // Cast pointer
            gr_complex * out = (gr_complex * ) output_items[0];

            // Create each output matrix
            for (int i = 0; i < output_matrices; i++) {
                // Form input matrix
                for (int k = 0; k < d_num_inputs; k++) {
                    memcpy((void * ) d_input_matrix.colptr(k), ((gr_complex * ) input_items[k] + i * d_nonoverlap_size),
                        sizeof(gr_complex) * d_snapshot_size);
                }

                // Make output pointer into matrix pointer
                arma::cx_fmat out_matrix(out + d_num_inputs * d_num_inputs * i, d_num_inputs, d_num_inputs, COPY_MEM, FIX_SIZE);

                // Do autocorrelation
                out_matrix = (1.0 / d_snapshot_size) * d_input_matrix.st() * conj(d_input_matrix);
                if (d_avg_method == 1)
                    out_matrix = 0.5 * out_matrix + (0.5 / d_snapshot_size) * d_J * conj(out_matrix) * d_J;

            }

            // Tell runtime system how many input items we consumed on
            // each input stream.
            consume_each(d_nonoverlap_size * output_matrices);

            // Tell runtime system how many output items we produced.
            return (output_matrices);
        }

    } /* namespace doa */
} /* namespace gr */
```

MUSIC_lin_array_impl.cc

```
#  ifdef HAVE_CONFIG_H
# include "config.h"
#  endif
# include < gnuradio / io_signature.h >
#include "MUSIC_lin_array_impl.h"

#  define COPY_MEM false // Do not copy matrices into separate memory
#  define FIX_SIZE true // Keep dimensions of matrices constant

```

```cpp

namespace gr {
    namespace doa {

        MUSIC_lin_array::sptr
        MUSIC_lin_array::make(float norm_spacing, int num_targets, int num_ant_ele, int pspectrum_len) {
            return gnuradio::get_initial_sptr(new MUSIC_lin_array_impl(norm_spacing, num_targets, num_ant_ele, pspectrum_len));
        }

        /*
         * The private constructor
         */
        MUSIC_lin_array_impl::MUSIC_lin_array_impl(float norm_spacing, int num_targets, int num_ant_ele, int pspectrum_len): gr::sync_block("MUSIC_lin_array",
                gr::io_signature::make(1, 1, sizeof(gr_complex) * num_ant_ele * num_ant_ele),
                gr::io_signature::make(1, 1, sizeof(float) * pspectrum_len)),
            d_norm_spacing(norm_spacing),
            d_num_targets(num_targets),
            d_num_ant_ele(num_ant_ele),
            d_pspectrum_len(pspectrum_len) {
                // form antenna array locations centered around zero and normalize
                d_array_loc = fcolvec(d_num_ant_ele, fill::zeros);
                for (int nn = 0; nn < d_num_ant_ele; nn++) {
                    d_array_loc(nn) = d_norm_spacing * 0.5 * (d_num_ant_ele - 1 - 2 * nn);
                }

                // form theta vector
                d_theta = new float[d_pspectrum_len];
                d_theta[0] = 0.0;
                float theta_prev = 0.0, theta;
                for (int ii = 1; ii < d_pspectrum_len; ii++) {
                    theta = theta_prev + 180.0 / d_pspectrum_len;
                    theta_prev = theta;
                    d_theta[ii] = datum::pi * theta / 180.0;
                }

                // form array response matrix
                cx_fcolvec vii_temp(d_num_ant_ele, fill::zeros);
                d_vii_matrix = cx_fmat(d_num_ant_ele, d_pspectrum_len);
                d_vii_matrix_trans = cx_fmat(d_pspectrum_len, d_num_ant_ele);
                for (int ii = 0; ii < d_pspectrum_len; ii++) {
                    // generate array manifold vector for each theta
                    amv(vii_temp, d_array_loc, d_theta[ii]);
                    // add as column to matrix
                    d_vii_matrix.col(ii) = vii_temp;
                }
                // save transposed copy
                d_vii_matrix_trans = trans(d_vii_matrix);

            }

        /*
         * Our virtual destructor.
         */
        MUSIC_lin_array_impl::~MUSIC_lin_array_impl() {}

        // array manifold vector generating function
        void MUSIC_lin_array_impl::amv(cx_fcolvec & v_ii, fcolvec & array_loc, float theta) {
            // sqrt(-1)
            const gr_complex i = gr_complex(0.0, 1.0);
            // array manifold vector
            v_ii = exp(i * (-1.0 * 2 * datum::pi * cos(theta) * array_loc));
        }

        int
        MUSIC_lin_array_impl::work(int noutput_items,
            gr_vector_const_void_star & input_items,
            gr_vector_void_star & output_items) {
            const gr_complex * in = (const gr_complex * ) input_items[0];
            float * out = (float * ) output_items[0];

            // process each input vector (Rxx matrix)
            fvec eig_val;
            cx_fmat eig_vec;
            cx_fmat U_N;
            cx_fmat U_N_sq;
            for (int item = 0; item < noutput_items; item++) {
                // make input pointer into matrix pointer
                cx_fmat in_matrix( in +item * d_num_ant_ele * d_num_ant_ele, d_num_ant_ele, d_num_ant_ele);
                fvec out_vec(out + item * d_pspectrum_len, d_pspectrum_len, COPY_MEM, FIX_SIZE);

            // determine EVD of the auto-correlation matrix
            eig_sym(eig_val, eig_vec, in_matrix);

            // noise subspace and its square matrix
            U_N = eig_vec.cols(0, d_num_ant_ele - d_num_targets - 1);
            U_N_sq = U_N * trans(U_N);

            // determine pseudo-spectrum for each value of theta in [0.0, 180.0)
            gr_complex Q_temp;
            for (int ii = 0; ii < d_pspectrum_len; ii++) {

                  Q_temp = as_scalar(d_vii_matrix_trans.row(ii) * U_N_sq * d_vii_matrix.col(ii));
                  out_vec(ii) = 1.0 / Q_temp.real();
                }
                out_vec = 10.0 * log10(out_vec / out_vec.max());

              }

              // Tell runtime system how many output items we produced.
              return noutput_items;
            }

      } /* namespace doa */
    } /* namespace gr */
```

Planned Course of Action

Run the test for direction-finding project from coherence-receiver website.

Resources and relevant Forum Posts

[1] “RTL-SDR Blog silver dongle first impressions, compared to NooElec blue dongle”https://medium.com/@rxseger/rtl-sdr-blog-silver-dongle-first-impressions-compared-to-nooelec-blue-dongle-4053729ab8c7
[2] “VIDEO TUTORIAL: INSTALLING GQRX AND RTL-SDR ON A RASPBERRY PI”https://www.rtl-sdr.com/video-tutorial-installing-gqrx-and-rtl-sdr-on-a-raspberry-pi/
[3] “DIGITAL RADIO WITH A RASPBERRY PI” http://www.michaelcarden.net/?p=48
[4] “Guided Tutorial GNU Radio in Python
”https://wiki.gnuradio.org/index.php/Guided_Tutorial_GNU_Radio_in_Python
[5] “GRCon17 - Real-Time Direction Finding Using Two Antennas on an Android Phone - Sam Whiting” https://www.youtube.com/watch?v=jptYYiHth8U
[6]” Real-Time Direction Finding Using Two Antennas on an Android Phone - Sam Whiting”

https://www.gnuradio.org/wp-content/uploads/2017/12/Todd-Moon-Gnuradio-DOA.pdf
[7]"gnuradio-doa" https://github.com/samwhiting/gnuradio-doa/tree/master/flowgraphs
[8]" GNURadioCompanion" https://wiki.gnuradio.org/index.php/GNURadioCompanion
[9] "Install RTL-SDR on Raspberry PI"
https://gist.github.com/floehopper/99a0c8931f9d779b0998