Introduction to GNUradio, Software Defined Radio, and Digital Signal Processing

Rhys Lockard

1 Introduction

Since the development of SDR or Software Defined Radio, signal processors have began to move away from using analog devices to using a computer with SDR installed, in which the signal is processed in a similar fashion as with an Analog but with the ability of adapting the software to the needs of the user.

For my research purpose; GNU-radio was the preferred software.

2 Linux and GNU-radio

In order for me to utilize GNU-radio it was necessary to install Linux. This was my first introduction the operating system and I ended up having a few simple issues.

2.1 Installing Linux

Getting Linux installed was not very difficult. I chose Linux Mint as the desktop is most similar to that of windows. With that I created a bootable USB drive with windows and installed Linux Mint as a dual boot with windows 10. This way allows for the seamless switch back and forth between the two operating systems.

After getting Linux installed and booting it up is where I came into issues. When I did the install, Linux Mint had issues with the current Nvidia driver, and as a result when I sent the graphics through the video card as opposed to the on board graphics on the processor, on login I would just have a black screen with a blinking cursor in the top left-hand side. In order to fix this; I had to do a complete wipe of both windows 10 and Linux Mint, Reinstall both, and then go to Nvidia website and found a corrected driver. Upon making those changes I was able to boot up into Linux Mint

2.2 installing GNU-radio

This was done by following the directions given on http://wvurail.org/dspira/labs/01/#11-installation-guide. I followed the guide to the letter and had absolutely no issues with installing GNU-radio.

2.3 Installing RTL-SDR

The RTL-SDR is the device that connects to your computer via USB and allows the user to receive radio signals from an antenna. It does this by allowing the transfer of raw I/Q samples to the computer.

The device that I chose pictured above is the RTL-SDR v3. This device was designed specifically with SDR needs in mind and is cheaper compared to Airspy or the FlightAware dongle. The RTL-SDR v3 has a bandwidth of 2.4 MHz with a frequency range of 500 kHz – 1766MHz. And since it will be used for detecting neutral Hydrogen that frequency range was satisfactory.

In order to install the device I followed instructions given by https://osmocom.org/projects/sdr/wiki/rtl-sdr#Credits. Open a terminal in Linux and enter the following commands:

```
cd rtl-sdr/
mkdir build
cd build
cmake ../
make
sudo make install
sudo ldconfig
```

```
cmake ../ -DINSTALL_UDEV_RULES=ON
```

```
cd rtl-sdr/
autoreconf -i
./configure
make
sudo make install
sudo ldconfig
```

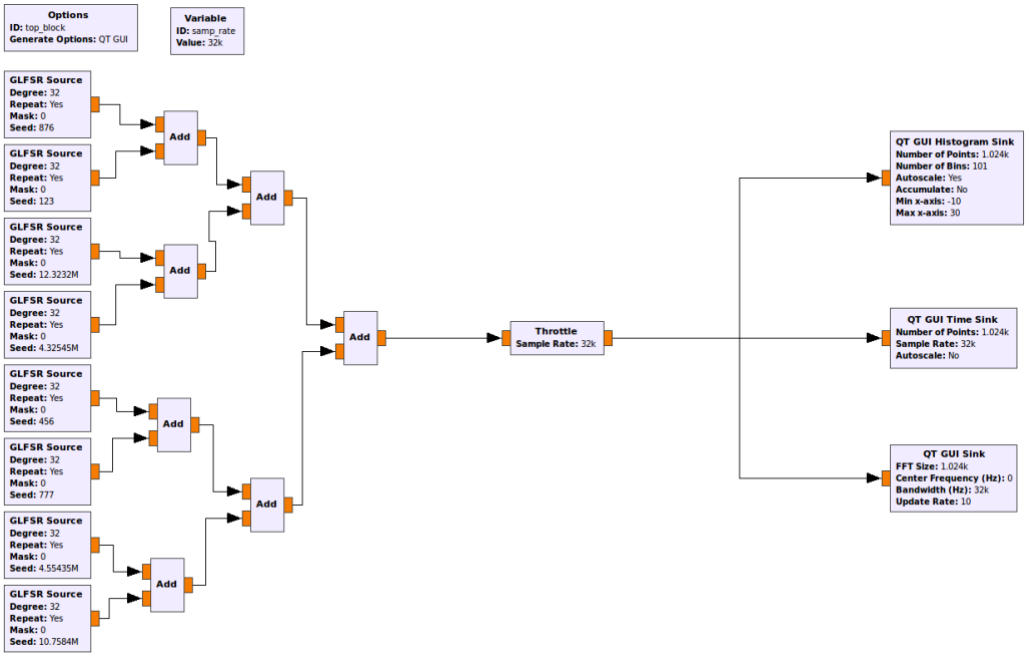
```
sudo make install-udev-rules
```

Once those instructions have been followed my computer was able to access the dongle.

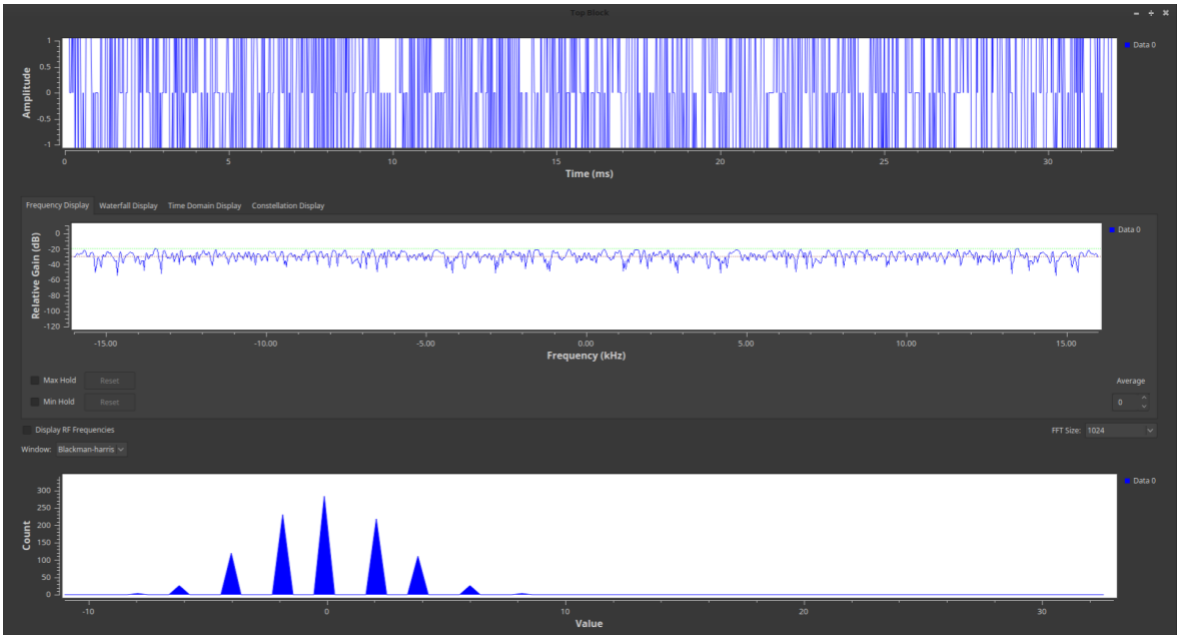
2.4 Using GNU-radio

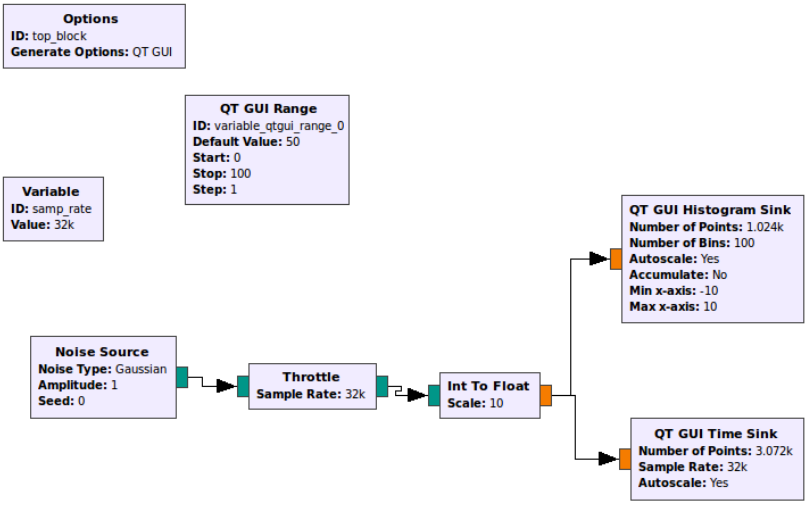
After Installing GNU-radio I began to mess around with it. I put together various flow graphs to help practice and understand Digital Signal Processing. With that I built; Gaussian Noise block, Histogram, Waveform generators, and a FM-Radio.

Gaussian Noise Block

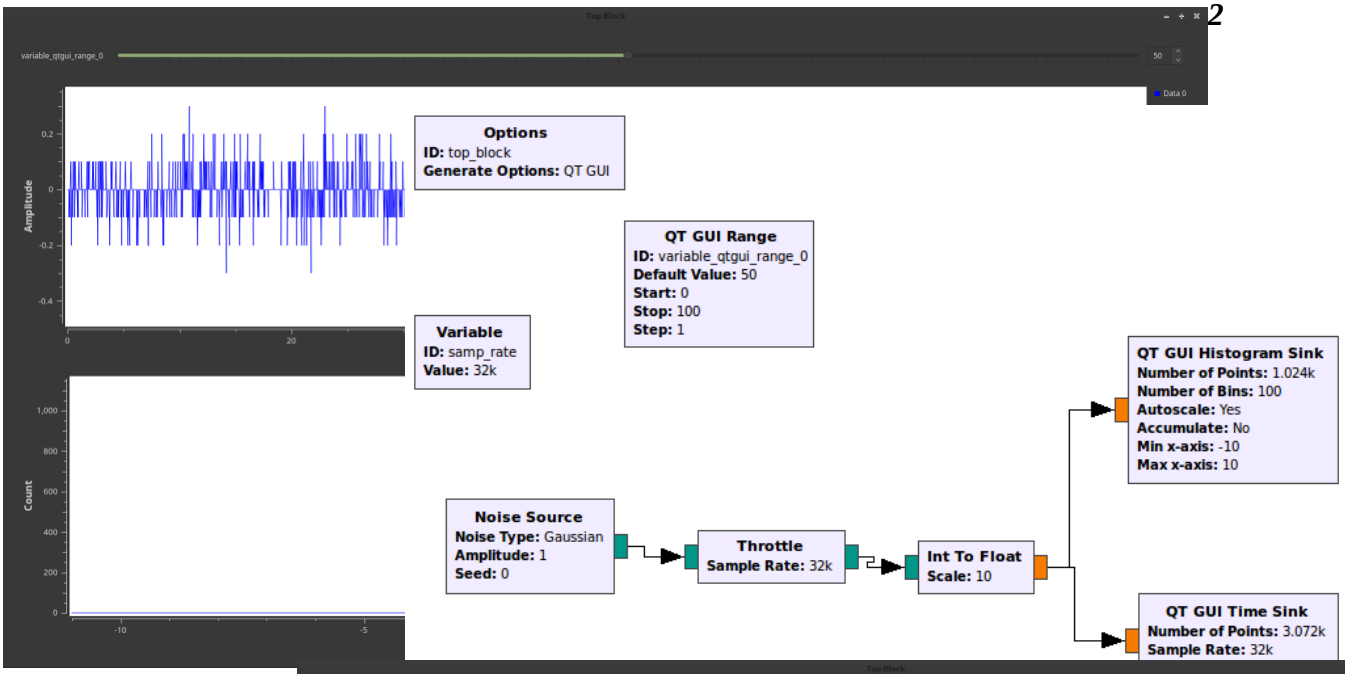


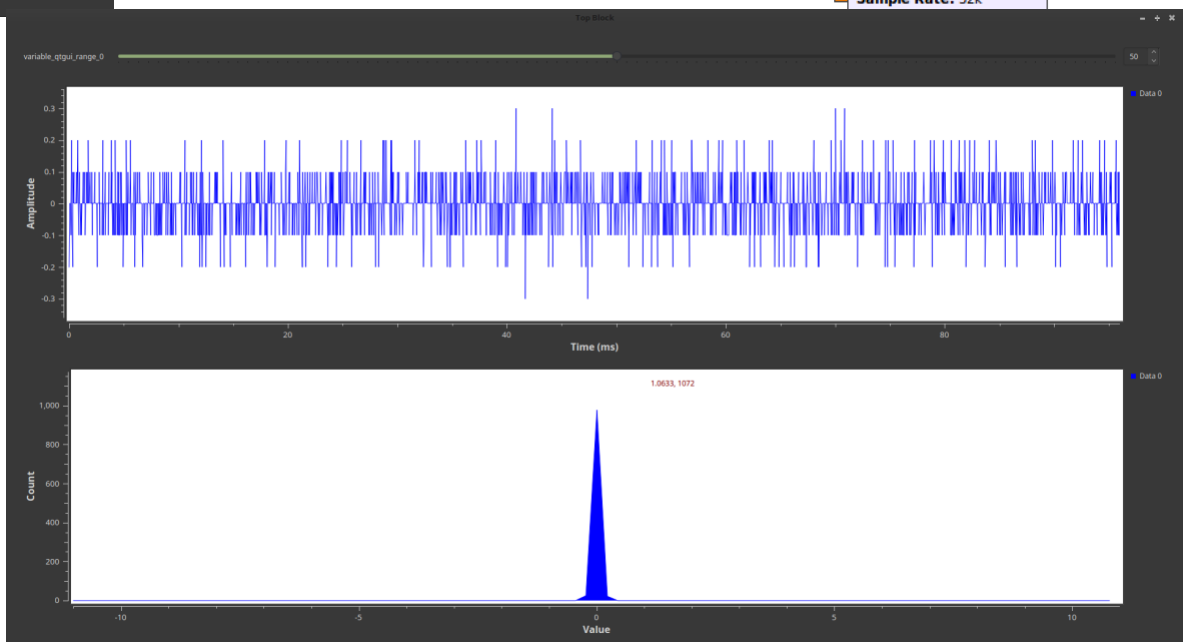
Histogram 1



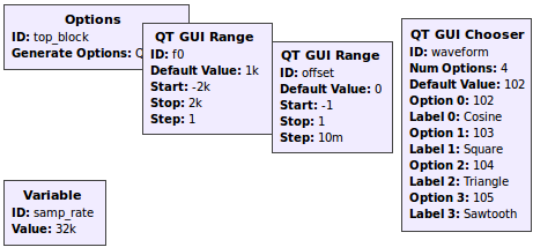

Options
ID: top_block
Generate Options: QT GUI
QT GUI Range
ID: variable_qtgui_range_0
Default Value: 50
Start: 0
Stop: 100
Step: 1
Variable
ID: samp_rate
Value: 32k
QT GUI Histogram Sink
Number of Points: 1.024k
Number of Bins: 100
Autoscale: Yes
Accumulate: No
Min x-axis: -10
Max x-axis: 10
Noise Source
Noise Type: Gaussian
Amplitude: 1
Seed: 0
Throttle
Sample Rate: 32k
Int To Float
Scale: 10
QT GUI Time Sink
Number of Points: 3.072k
Sample Rate: 32k
Autoscale: Yes

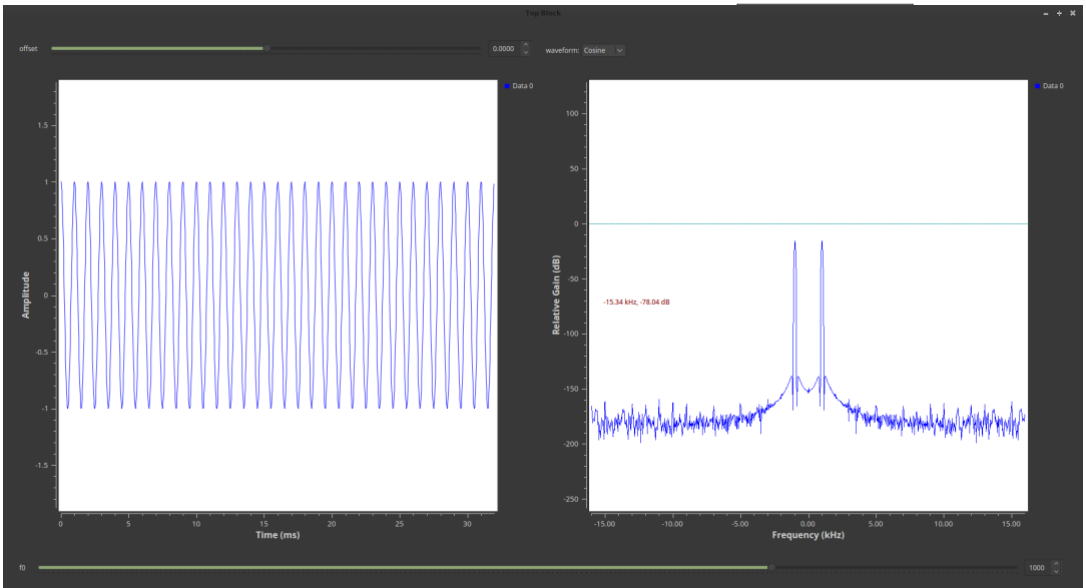
Histogram 2


Top Block
variable_qtgui_range_0
Data 0
Amplitude
Count
Options
ID: top_block
Generate Options: QT GUI
QT GUI Range
ID: variable_qtgui_range_0
Default Value: 50
Start: 0
Stop: 100
Step: 1
Variable
ID: samp_rate
Value: 32k
QT GUI Histogram Sink
Number of Points: 1.024k
Number of Bins: 100
Autoscale: Yes
Accumulate: No
Min x-axis: -10
Max x-axis: 10
Noise Source
Noise Type: Gaussian
Amplitude: 1
Seed: 0
Throttle
Sample Rate: 32k
Int To Float
Scale: 10
QT GUI Time Sink
Number of Points: 3.072k
Sample Rate: 32k


Top Block
variable_qtgui_range_0
Data 0
Amplitude
Time (ms)
Count
Value

Waveform Generator 1





Waveform Generator 2

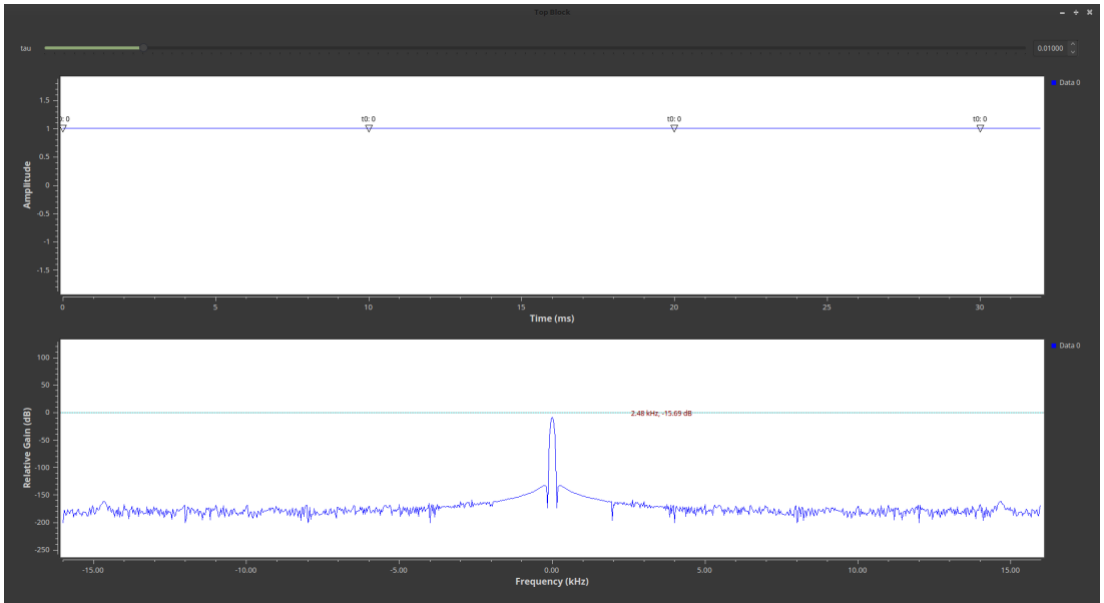
Options
ID: top_block
Generate Options: QT GUI

QT GUI Range
ID: tau
Default Value: 10m
Start: 0
Stop: 100m
Step: 1m

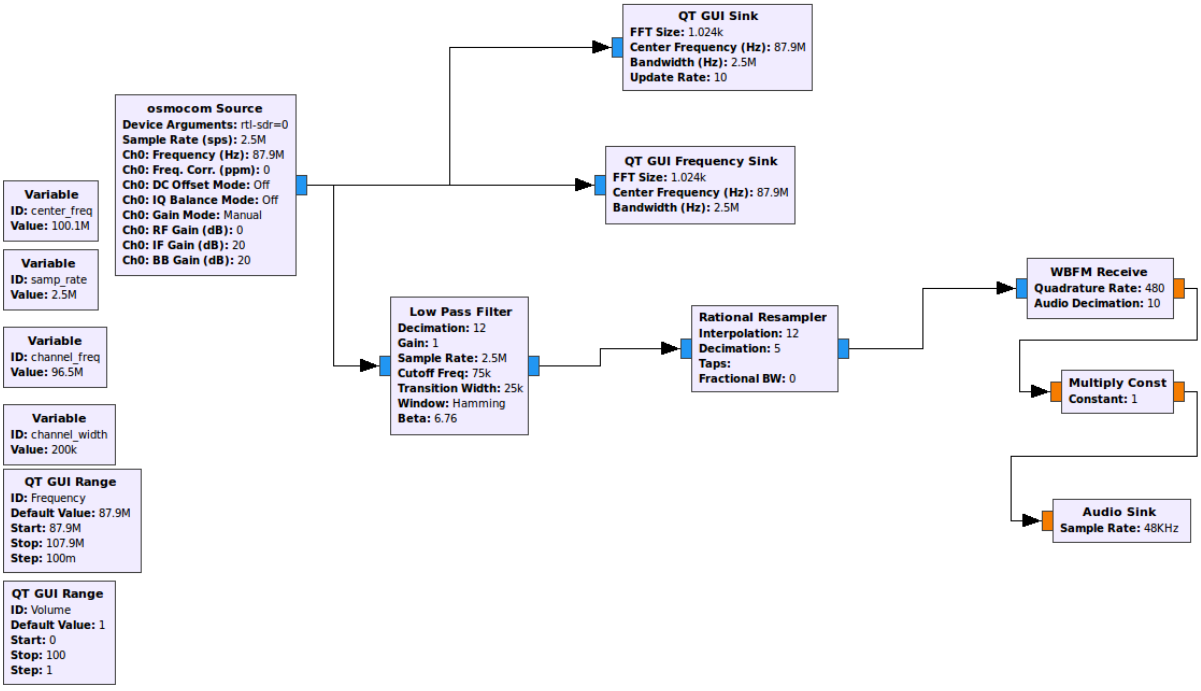
Tag Object
ID: tag
Offset: 0
Key: t0
Value: 0
Source ID: vecsrc

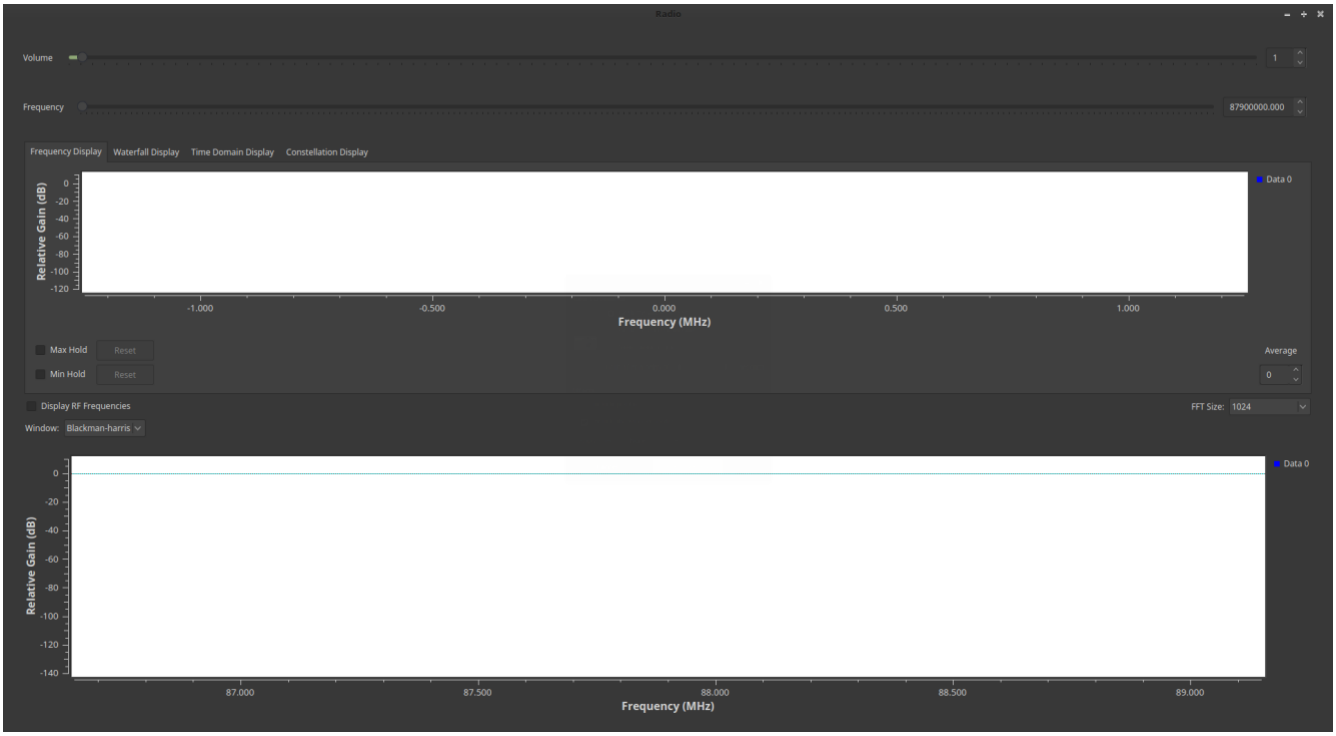
Import
Import: np

Variable
ID: samp_rate
Value: 32k



FM-Radio

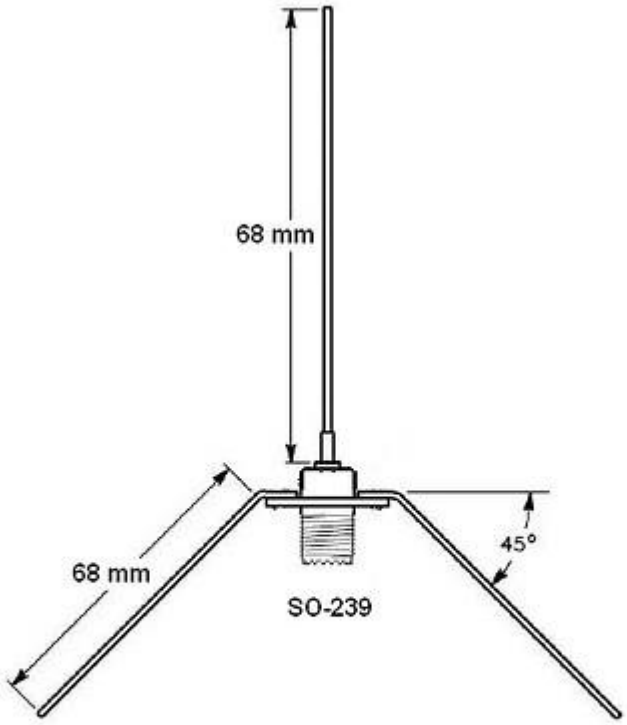

QT GUI Sink
FFT Size: 1.024k
Center Frequency (Hz): 87.9M
Bandwidth (Hz): 2.5M
Update Rate: 10
osmocom Source
Device Arguments: rtl-sdr=0
Sample Rate (sps): 2.5M
Ch0: Frequency (Hz): 87.9M
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 20
Ch0: BB Gain (dB): 20
QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 87.9M
Bandwidth (Hz): 2.5M
Variable
ID: center_freq
Value: 100.1M
Variable
ID: samp_rate
Value: 2.5M
Variable
ID: channel_freq
Value: 96.5M
Variable
ID: channel_width
Value: 200k
QT GUI Range
ID: Frequency
Default Value: 87.9M
Start: 87.9M
Stop: 107.9M
Step: 100m
QT GUI Range
ID: Volume
Default Value: 1
Start: 0
Stop: 100
Step: 1
Low Pass Filter
Decimation: 12
Gain: 1
Sample Rate: 2.5M
Cutoff Freq: 75k
Transition Width: 25k
Window: Hamming
Beta: 6.76
Rational Resampler
Interpolation: 12
Decimation: 5
Taps:
Fractional BW: 0
WBFM Receive
Quadrature Rate: 480
Audio Decimation: 10
Multiply Const
Constant: 1
Audio Sink
Sample Rate: 48KHz



3 Antenna

3.1 Design Choice

The next stage of my research came to finding a simple antenna design and building it. For my purposes I chose a Spider Antenna. They are typically more efficient than a normal monopole and since it would be used for receiving airplane data I wanted it to be as efficient as possible



SPIDER ANTENNA

Ground Plane formed by 4, 6 or 8 Slanting Radials

3.2 Calculations

The Legs and the main receiving antenna length were calculated from the equation

$\lambda = C/f,\ C = 3.0*10^8$ m/s $f = 1.09*10^9$ MHz

$\lambda = 65.43$ MM

$$D = \frac{1}{\frac{1}{4\pi}\int_0^{2\pi}\int_0^{\pi} |F(\theta,\phi)|^2 \sin\theta d\theta d\phi}$$

D (θ,φ) = 1.5

$D_{db} = 10Log_{10}(\boldsymbol{D(\theta,\varphi)}$

$\boldsymbol{D_{db} = 1.76\ DB}$

$$\sqrt{\frac{\pi f \mu_o}{\sigma}} = R_s \quad f = 1.09*10^9\ \text{MHz} \quad \mu_o = 4\pi*10^{-7}\ \text{N/A}^2$$

$\boldsymbol{R_s = .00861\ \Omega}$

$$R_s\left(\frac{L}{(2\pi A)}\right) = R_{ohmic} \quad L = .068\ \text{m} \quad A = .00259\ \text{m}$$

$\boldsymbol{R_{ohmic} = .03599\ \Omega}$

$$\frac{2\pi n}{3}\left(\frac{\Delta L}{\lambda}\right) = R_r \qquad n = \omega\varepsilon \quad \varepsilon = 8.85*10^{-12} \qquad \omega = 2\pi f$$

$\boldsymbol{n = .606}$ $\qquad$ $\boldsymbol{\omega = 6.85*10^9}$

$\boldsymbol{R_r = .03135\ \Omega}$

$$\frac{R_r}{(R_r + R_{ohmic})} = \boldsymbol{E_r} \qquad \boldsymbol{E_r = .4655\ or\ 47\%}$$

3.3 Building

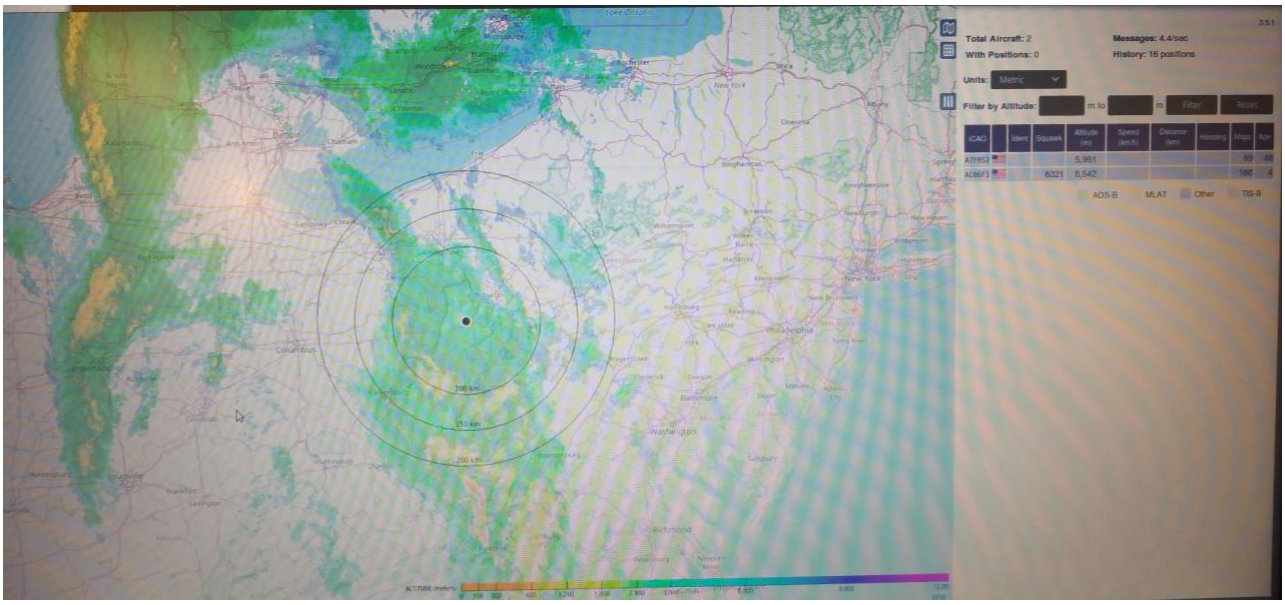
Putting the antenna together took some finesse. Especially the legs, the solder did not want to stick to the aluminum casing on the S0-329 connector. After some clamping and maneuvering I was able to get them all in their proper positions

4 Dump1090 and RTL-SDR

4.2 Installing Dump1090

The flight tracking software that I used is Dump1090. And following the install instructions given at https://github.com/MalcolmRobb/dump1090. I went with Malcolm Robb's version of dump1090 as his program has some applications with Flight Aware; a website for amateur aircraft enthusiasts to submit their data.

4.3 Using Dump1090



After getting everything properly installed and set-up I began to collect data using my antenna, the RTL-SDR v3 and Dump1090.

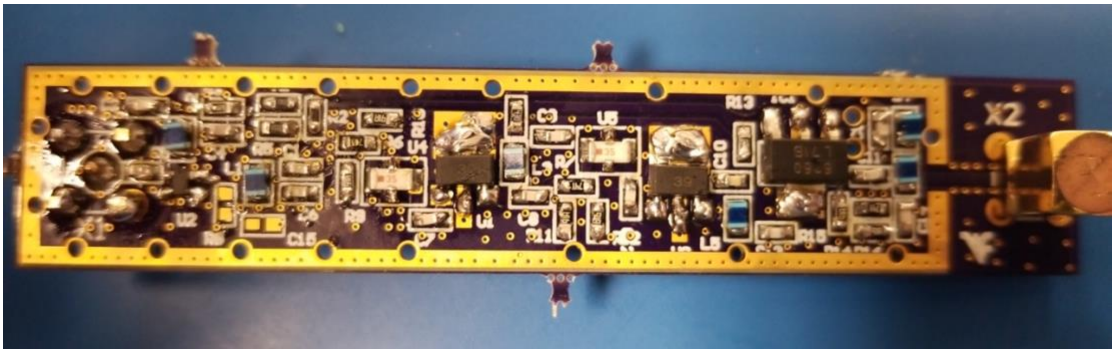
I collected data for a couples days and wanted to be able to see it visually. I discovered https://github.com/aarkebauer/dump1090plot. This a program written by aarkebauer, in which it takes the data from Dump1090 and then plots a 3D map in which you can now see the paths of the planes while they were within range of my antenna. Though I could not get it to work it was a good learning experience.

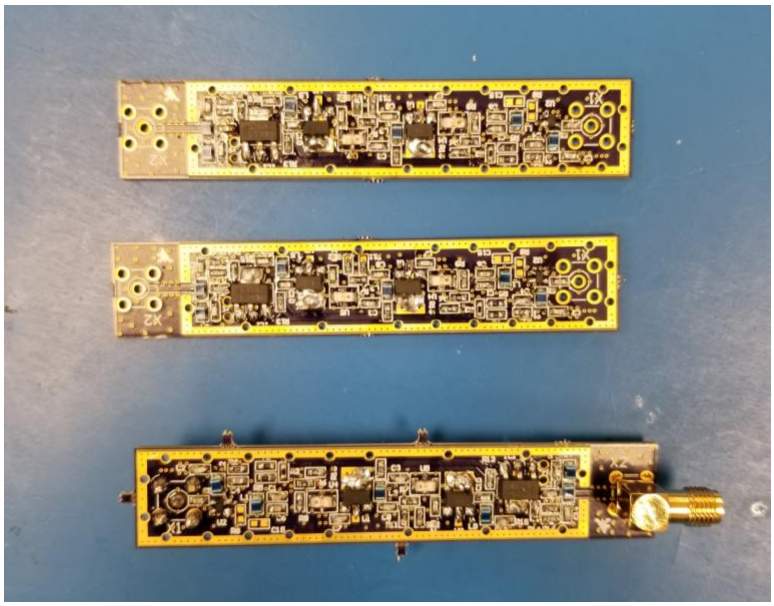
5 Radio Observatory

The next stage in my research was doing some actual astronomical readings. The goal being to make a spectrometer in GNUradio and build a LNA (Low Noise Amplifier) and then use those devices to inspect the 21cm band of neutral Hydrogen that sits at 1420 MHz frequency.

5.1 LNA

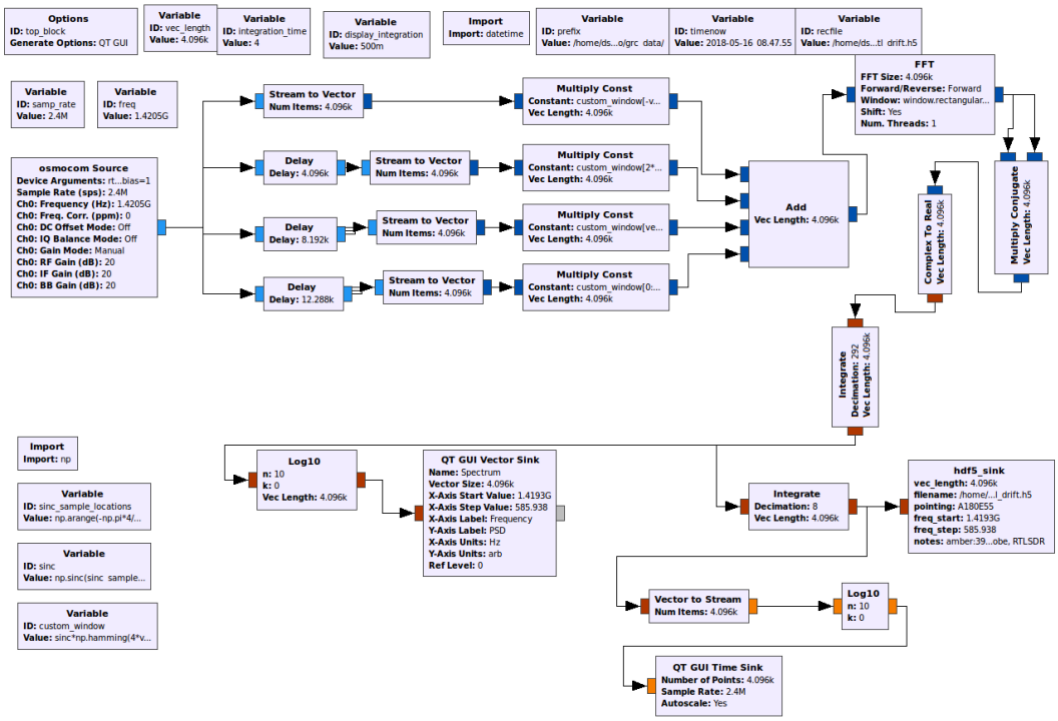
The LNA was designed by Dr. Kevin Bandura, the Faculty Advisor for WVU Radio Astronomy Instrumentation Lab. This was my first experience with this kind of soldering and working with components that small but it turned out well.





5.2 Spectrometer

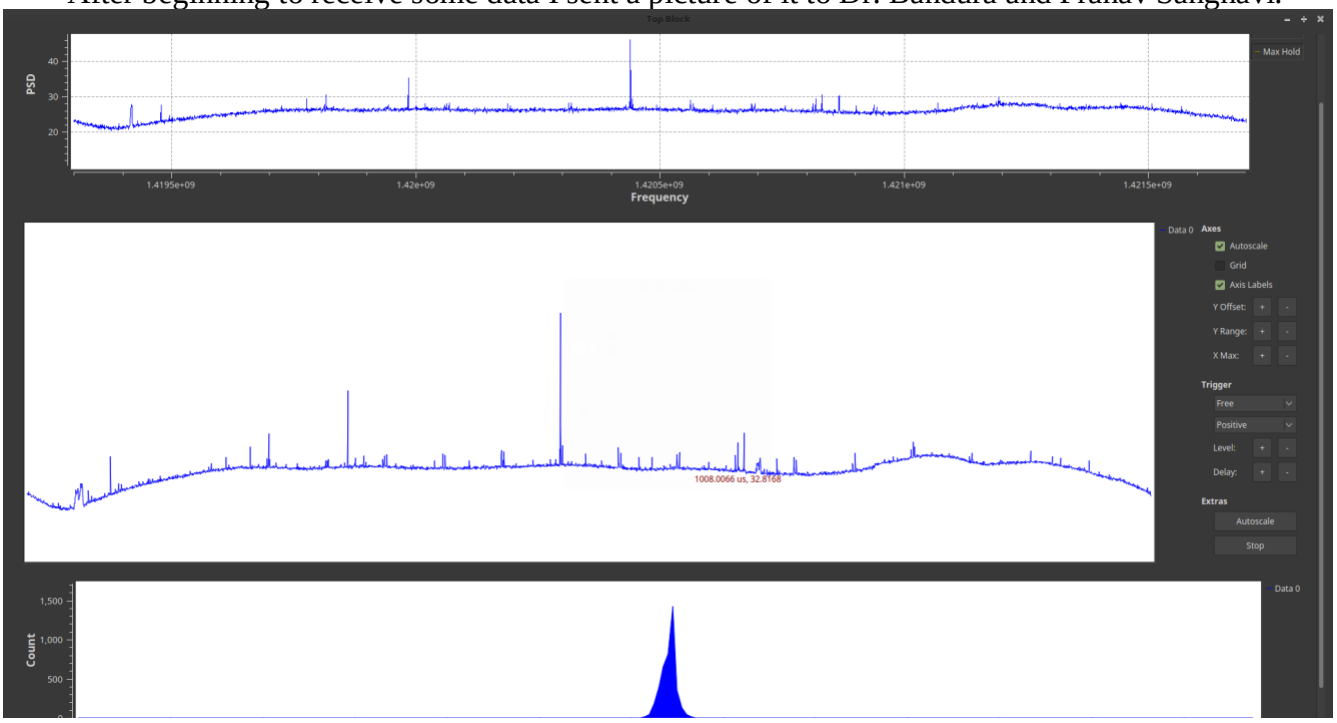
The spectrometer was an area where I had a lot of difficulty. I ended up using the spectrometer written by Dr. Bandura and is available at https://github.com/WVURAIL/dspira/tree/master/grc-flowgraphs. Below is the flow-graph for the spectrometer.



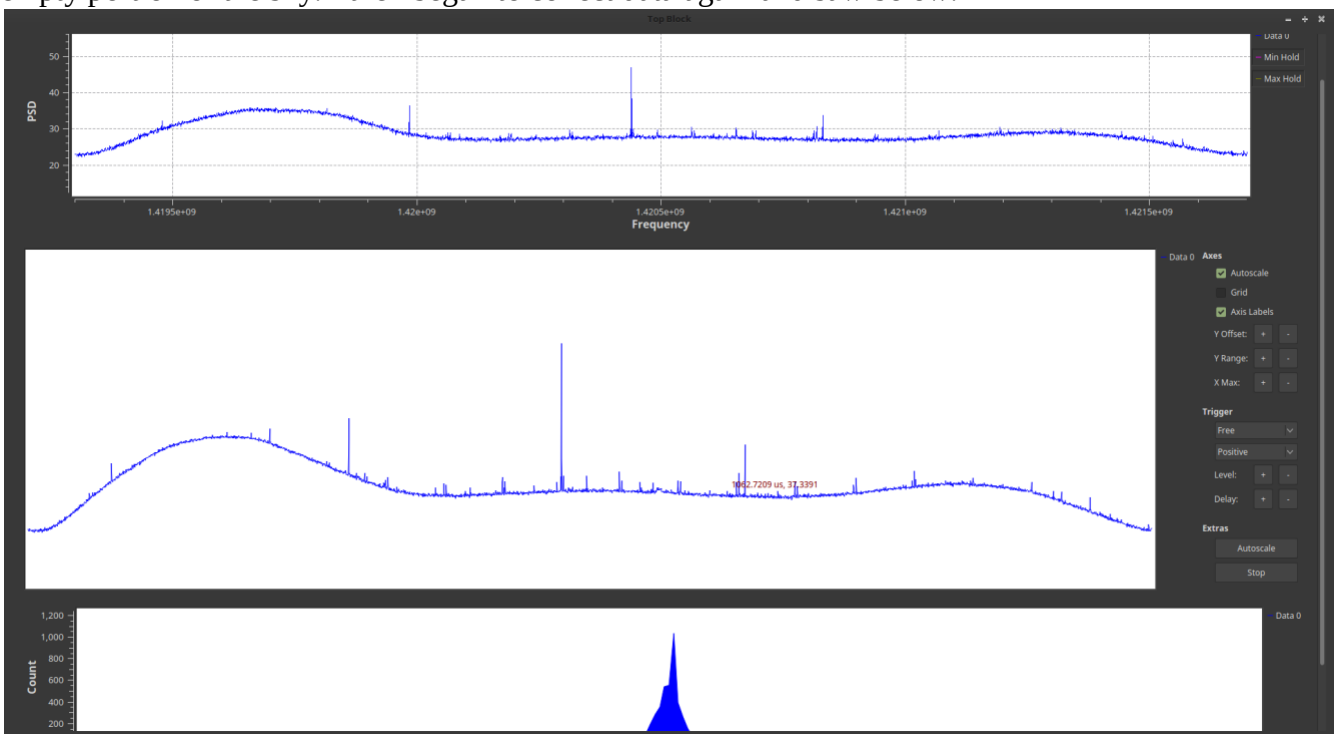
5.2 Data Collection

Now that I had all the components it was time to begin collecting data. Taking A horn antenna and pointing it to the Galactic Center of the Milky Way in which the 21cm neutral Hydrogen is most dense.

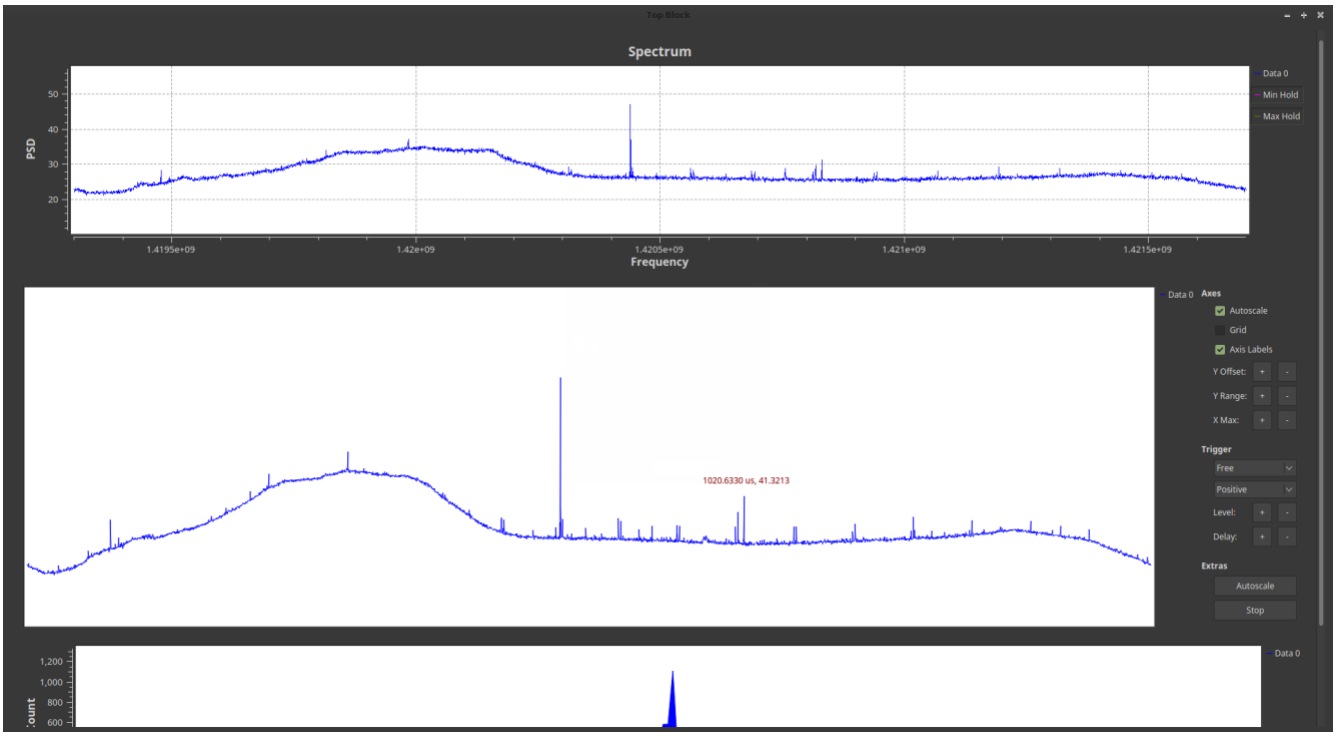
After beginning to receive some data I sent a picture of it to Dr. Bandura and Pranav Sanghavi.



In which they told me that was not how it was supposed to look and that there must have been something wrong. I re calibrated the system by pointing the horn at the ground and then pointing it an empty portion of the sky. I then began to collect data again and saw below.



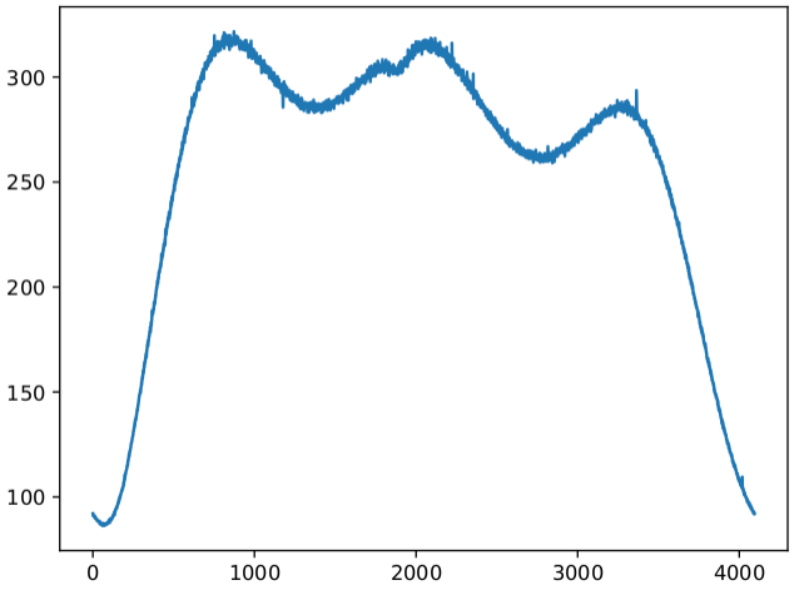
As the data appeared to be the same we knew something was wrong. I went through and changed the SMA cables that connected the LNA to the Horn and then to the V3 dongle. And then re-calibrated and collected data again.



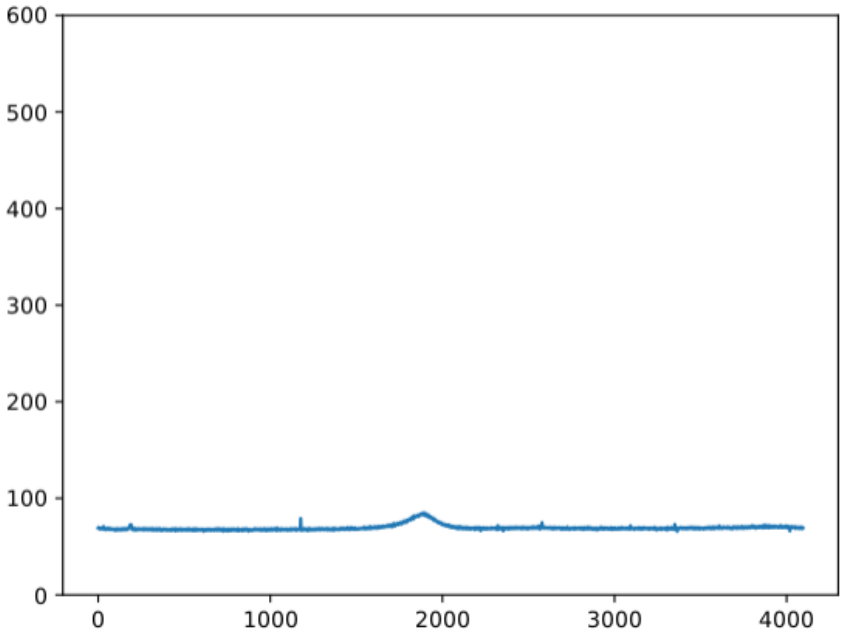
Again the data was incorrect. This time we decided that there must be something wrong with the LNA. Dr. Bandura began doing some testing and discovered there must be an issue with the resistance and capacitance on the board. With this we changed 4 resistors and 2 capacitors and the data came in as expected.

Calibration:

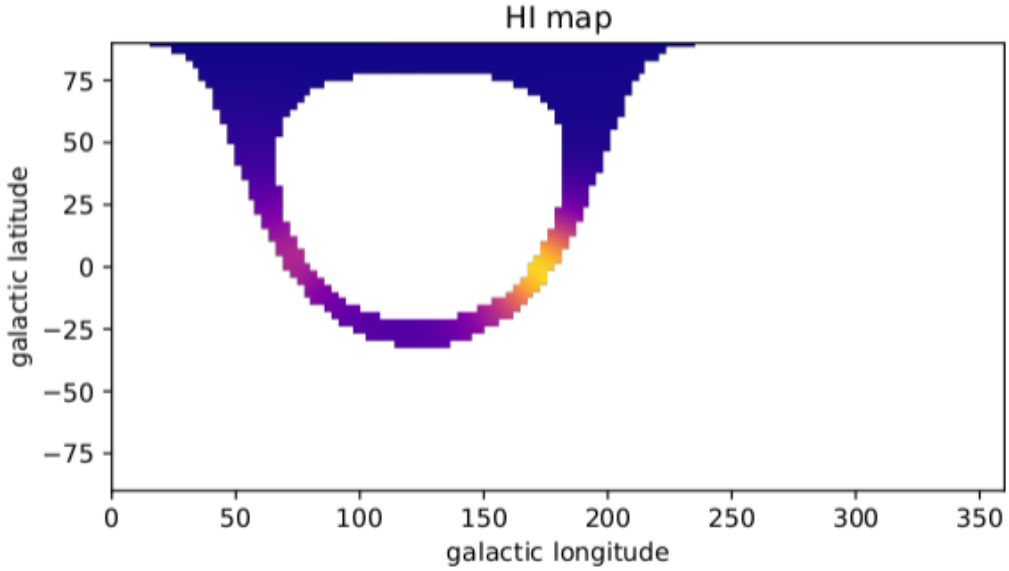
Gain Map



Temperature map



Galactic Map



6 Conclusion

The WVURAIL DSPIRA project was an awesome learning experience and gave light to what I would like to do as a career. The labs that Pranav had prepared at http://wvurail.org/dspira/labs/ were the most helpful to completing this project.

7 Code

aarkebaeur Plot Code

```
    #!/user/bin/env python2
import re, os
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def main():
    createfiles()   # takes the full data3.json and breaks it down into a single file for each
                        # plane (based off of icao hex code)
    plot()              # pull data from each file and make a 3d plot of each plane's path based
                        # on the latitude, longitude, and altitude data they contain

def createfiles():
    dat = open("data3.json", 'rU');

    flag = { }  # create dictionary to be used to store hex codes. If a hex code has already been
stored,
                # then the data file must exist for that code, so don't write it, just append to it
    i = 0;          # keys for dictionary (arbitrary, serve no purpose)

    for line in dat: # read source code

        hexdat = re.search('("hex":")([\w|\d]*)', line);
        if hexdat:
            hexdata = hexdat.group(2);
            flightdat = re.search('("flight":")([\w|\d]*)', line);
            if flightdat:
                flightdata = flightdat.group(2)
            latdat = re.search('("lat":)([-]*[\d]*\.[\d]*)', line);
            latdata = latdat.group(2)
            londat = re.search('("lon":)([-]*[\d]*\.[\d]*)', line);
            londata = londat.group(2)
            altdat = re.search('("altitude":)(\d*)', line);
            altdata = altdat.group(2)
            trackdat = re.search('("track":)(\d*)', line);
            trackdata = trackdat.group(2)
            speeddat = re.search('("speed":)(\d*)', line);
            speeddata = speeddat.group(2)
```

```
                filename = "data3" + ".json"

                if str(hexdata) in flag.values():     # If a hex code has already been stored, then the
data file
                                                                # must exist for that code, so don't write it,
just append
                        wrfile = open(filename, 'a');
                        wrfile.write("\n")
                else:
                        wrfile = open(filename, 'w');

                wrfile.write(hexdata + "\n");
                if flightdat:
                        wrfile.write(flightdata + "\n");

                # if latitude, longitude, or altitude data is a 0 (i.e. not present), don't write
                # any of the three - this cleans up the lists to eliminate bad data points when plotting
                if float(latdata) != 0:
                        if float(londata) != 0:
                                if float(altdata) != 0:
                                        wrfile.write(latdata + "\n");
                                        wrfile.write(londata + "\n");
                                        wrfile.write(altdata + "\n");
                else:
                        wrfile.write("\n");
                        wrfile.write("\n");
                        wrfile.write("\n");
                wrfile.write(trackdata + "\n");
                wrfile.write(speeddata + "\n");
                wrfile.close()

                flag[i] = str(hexdata) # update dictionary
                i +=1

def plot():

        ##############################################################################
        # This is where data is extracted from files and put into lists for plotting
        ##############################################################################

        allalts = { }
        alllats = { }
        alllons = { }
        allflights = { }
        counter = 0

        for file in os.listdir("/home/rhys"): # If an empty file named .DStore or something exists, it may
not work - ?
```

```
filename = "data3" + ".json";
workingfile = open(filename, 'rU')


worker = open(filename, 'rU');     # must open the file under a different name to count number of lines
length = 0;
for line in worker:  # count number of lines in file to determine how long lists (below) should be
    length +=1;


flightnumberlines = [x*8+1 for x in range(0, length)]
latitudelines = [x*8+2 for x in range(0, length)]
longitudelines = [x*8+3 for x in range(0, length)]
altitudelines = [x*8+4 for x in range(0, length)]

# initialize these as the empty list
latlist = list()
longlist = list()
altlist = list()
flightnumberlist = list()

i = 0
for line in workingfile:
    if i in latitudelines:
        latlist.append(line[0:len(line)-1])
    if i in longitudelines:
        longlist.append(line[0:len(line)-1])
    if i in altitudelines:
        altlist.append(line[0:len(line)-1])
    if i in flightnumberlines:
        flightnumberlist.append(line[0:len(line)-1])
    i += 1

#print altlist
#print latlist
#print longlist
#print flightnumberlist

# remove all null elements from the lists (planes that produced 0 complete position data points)
altlist = map(float, filter(None, altlist))
latlist = map(float, filter(None, latlist))
longlist = map(float, filter(None, longlist))
flightnumberlist = filter(None, flightnumberlist)

allalts[counter] = altlist
alllats[counter] = latlist
```

```
        alllons[counter] = longlist
        allflights[counter] = flightnumberlist

        counter += 1

        workingfile.close()
        worker.close()

    #print allalts
    #print alllats
    #print alllons
    #print allflights



    ################################################################################
############################
    ################################################################################
############################
    #                                                THIS IS WHERE PLOTTING OCCURS
                                    #
    ################################################################################
############################
    ################################################################################
############################



    ########################################################################
    #                    THESE VALUES ARE USED TO FORMAT THE PLOTS                 #
    ########################################################################

    # SET THESE LINES TO CURRENT LATITUDE AND LONGITUDE TO CENTER GRAPH
    currentlatitude = 0.0000
    currentlongitude = -0.0000
    # If the following is set to True it will plot from z = 0
    # If set to False it will plot from z = [minimum altitude] - 1000 ft
    plot_from_ground_level = False

    # The following is used to only plot flights under a certain altitude
    # Set this to float('inf') in order to see all flights
    only_plot_flights_under_this_altitude = float('inf') # to plot all: float('inf')

    # Setting this to True prints out the flight numbers for each plotted line
    # Setting this to False does not display flight numbers
    display_flight_number_labels = True

    # Set the following to True to plot planes WITHOUT flight number data
    # Set it to False to plot ONLY those planes WITH flight number data
```

```
plot_planes_without_flight_numbers = False

# If set to True planes with flight numbers containing only numbers
# (e.g. 1653) will be displayed. If set to False then ONLY flight
# numbers beginning with letters (e.g. UAL1653) will be displayed
plot_planes_with_number_only_flight_numbers = True

# Values used to change plot animation parameters are defined below #

# If set to True all flight numbers will be printed to a document
# titled flight_numbers.dat. If false, it will not make this file
print_flight_number_file = False

#########################################################################
#########################################################################
#########################################################################



plt.ion() # this is necessary for animating (spinning in this case) the plot - it must stay here!!!

fig = plt.figure(figsize=(15,9)) # this figure size fits well to macbook pro 13.3" screen
ax = fig.add_subplot(111, projection='3d')



minimumaltitude = filter(None, allalts.values())
minimums = list() # initialize list of minimum altitudes of each plane
for item in minimumaltitude:
    minimums.append(min(item))
minimumaltitude = min(minimums) # find the absolute lowest altitude to be used as the bottom
of the graph

if plot_from_ground_level != True: # Described above, used to either plot from 0 up or [minalt -
1000] up
    floor = minimumaltitude - 1000
else:
    floor = 0



# these 2 circles are used to give an idea of the locations of the planes - each is ~120 miles in
diameter
xcircle = range(-100,101)
xcircle = [float(item) / 100 for item in xcircle] # create a list from -2 to 2 with increment .1
ycirclepos = [pow(pow(1,2) - (pow(item, 2)), .5) for item in xcircle]
ycircleneg = [-pow(pow(1,2) - (pow(item, 2)), .5) for item in xcircle]
ax.plot(xcircle, ycirclepos, floor, color='0.7')
```

```
    ax.plot(xcircle, ycircleneg, floor, color='0.7')
    # place the 120mi label at a 45 degree angle (in the NE quadrant of the plot)
    ax.text(-((pow(2,0.5)/2)-.05),(pow(2,0.5)/2)-.05,floor,"60 mi", fontsize=8, color='0.7')

    xcircle2 = range(-200,201)
    xcircle2 = [float(item) / 100 for item in xcircle2] # create a list from -2 to 2 with increment .1
    ycirclepos2 = [pow(pow(2,2) - (pow(item, 2)), .5) for item in xcircle2]
    ycircleneg2 = [-pow(pow(2,2) - (pow(item, 2)), .5) for item in xcircle2]
    ax.plot(xcircle2, ycirclepos2, floor, color='0.7')
    ax.plot(xcircle2, ycircleneg2, floor, color='0.7')
    # place the 120mi label at a 45 degree angle (in the NE quadrant of the plot)
    ax.text(-(pow(2,0.5)-.05),pow(2,0.5)-.05,floor,"120 mi", fontsize=8, color='0.7')


    all_plotted_lats = list()
    all_plotted_lons = list()
    all_plotted_alts = list()
    #feetperdegree = 362776; # to measure altitude in (roughly) degrees latitude
    for plot in range(len(allalts.values())):
        xx = alllats.values()[plot]
        xx = [item - currentlatitude for item in xx] # center the graph around current latitude
        xx = [-item for item in xx] # flip everything over the x axis so directions line up
                                    # Think about this like you're looking up at the planes
                                    # from below - positions of the cardinal directions are
swapped
                                    # compared to looking down from above as you are in
the plot
        yy = alllons.values()[plot]
        yy = [item - currentlongitude for item in yy] # center graph around current longitude
        zz = allalts.values()[plot]
        #zz = [item / feetperdegree for item in zz] # to measure altitude in (roughly) degrees
latitude

        # obtain the flight number, if any
        flightnum = allflights.values()[plot]
        if len(flightnum) != 0:
            # If the flight number is only a number (e.g. 1653), then plot/don't plot according to
setting
            if plot_planes_with_number_only_flight_numbers != True:
                chars = []
                for c in flightnum[0]:
                    chars.append(c)
                letters = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M",
"N", "O", "P",
                 "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]
                if chars[0] in letters:
                    flightnum = flightnum
                    if print_flight_number_file == True:
                        flight_num_file = open("flight_numbers.dat", 'a')
```

```
                    flight_num_file.write(flightnum[0] + "\n")
                else:
                    flightnum = " "
            else:
                flightnum = flightnum
                if print_flight_number_file == True:
                    flight_num_file = open("flight_numbers.dat", 'a')
                    flight_num_file.write(flightnum[0] + "\n")
        else:
            flightnum = " "



        ################
        # plot the data
        ################

        # x-axis : latitude (degrees from current location, as set below)
        # y-axis : longitude (degrees from current location, as set below)
        # z-axis : altitude (ft)

        if len(xx) != 0:                    # don't attempt to plot empty lists

            # if the flight's minimum altitude is under the set threshold, plot the flight
            if only_plot_flights_under_this_altitude >= min(allalts.values()[plot]):
                if plot_planes_without_flight_numbers != True: # this is set/described above
                    if flightnum != " ":                        # plot only planes WITH
flight numbers
                        ax.plot(xx, yy, zz)             # use ax.scatter(xx, yy, zz) for a scatter
plot, etc.

                        # The following statements keep track of all lats, lons, and alts that are
PLOTTED
                        # This is used for scaling the final plot, as described below
                        for item in xx:
                            all_plotted_lats.append(item)
                        for item in yy:
                            all_plotted_lons.append(item)
                        for item in zz:
                            all_plotted_alts.append(item)

                        if display_flight_number_labels == True:
                            ax.text(xx[0],yy[0],zz[0], " "+flightnum[0], fontsize=6)
                else:
                    ax.plot(xx, yy, zz)             # use ax.scatter(xx, yy, zz) for a scatter plot,
etc.
```

```
                    # The following statements keep track of all lats, lons, and alts that are
PLOTTED
                    # This is used for scaling the final plot, as described below
                    for item in xx:
                        all_plotted_lats.append(item)
                    for item in yy:
                        all_plotted_lons.append(item)
                    for item in zz:
                        all_plotted_alts.append(item)

                    if display_flight_number_labels == True:
                        ax.text(xx[0],yy[0],zz[0], " "+flightnum[0], fontsize=6)


    # *** all_plotted_lats, all_plotted_alts, all_plotted_lons contain only values of PLOTTED
planes *** #
    # The following obtains min and max latitude and longitude to be used in setting frame limits
    # of the plot - a makeshift way of scaling
    minimumlat =  min(all_plotted_lats)
    maximumlat =  max(all_plotted_lats)
    minimumlon =  min(all_plotted_lons)
    maximumlon =  max(all_plotted_lons)
    # The z-axis autoscale seems to be good enough, but these are here anyway for possible future
use
    minimumalt =  min(all_plotted_alts)
    maximumalt =  max(all_plotted_alts)

    #ax.set_zlim(bottom=floor)

    # to keep the graph centered, the x and y limits must all be the same. So, I chose the largest
    # relative lat OR lon to be the absolute limit so that every point will stay within the frame
    # note: the z autoscale seems to be good enough
    s = max(abs(minimumlat), abs(minimumlon), abs(maximumlat), abs(maximumlon))
    ax.set_xlim(left=-s, right=s)        # x is latitude
    ax.set_ylim(bottom=-s, top=s)            # y is longitude

    #plt.show() # this is used for a static graph




    # make lines for x and y axes that extend slightly past the 120mi (2 degrees) marking circle
    xlimits = ax.get_xlim()
    ylimits = ax.get_ylim()
    ax.plot([float(xlimits[0])-.5, float(xlimits[1]+.5)], [0,0], [floor, floor], color='0.7')
    ax.plot([0,0], [float(ylimits[0])-.5, float(ylimits[1]+.5)], [floor, floor], color='0.7')
```

```
    # label the cardinal directions
    ax.text(0, s+.55, floor, " E", color='0.7')
    ax.text(0, -s-.55, floor, " W", color='0.7')
    ax.text(s+.55, 0, floor, " S", color='0.7')
    ax.text(-s-.55, 0, floor, " N", color='0.7')

    zticks = ax.get_zticks() # get the values that are displayed on the z axis by default
    ax.plot([0,0], [0,0], [floor,max(zticks)], color='0.7') # line for z axis


    # this prints a the default z-axis labels next to the new, centered, z axis
    itemnumber = 0
    for item in zticks:
        if (itemnumber % 2 != 0): # display every other label from the default z-axis labels
            ax.text(0,0,item," " + str(int(item)), fontsize=8, color='0.7') # z axis drawn at x=0, y=0
        itemnumber+=1

    plt.axis('off')             # turn ALL axes off



    # this is to rotate the plot at an elevation of 30 degrees through a full 360 degrees azimuthally
    # it quits after rotating 360 degrees (or whatever the upper limit of the range is)

    ##########################################################################################
    ##########################################################################################
    # ANIMATION PARAMETERS:
    speed = 1.5 # This is the speed at which the plot rotates. *** Higher numbers = slower ***
    revolutions = 3 # This is the number of revolutions that the plot will complete
    ##########################################################################################
    ##########################################################################################

    azimuths = range(0,int(((360*revolutions)*speed)+1))
    azimuths = [float(item) / speed for item in azimuths]
    for azimuth in azimuths:
        ax.view_init(elev=45, azim=azimuth)
        plt.draw()




# This is the standard boilerplate that calls the main() function initially
if __name__ == '__main__':
```

main()

Reshape Data Code

```
    ############################
## Read and Reshape Data
############################

specl = np.fromfile("/media/rhys/Radio.Observatory/signal1.dat", dtype=np.float32) #open data in read mode and put all the data as a numpy array spec
specl.shape # .shape shoews the array dimension.shape
spec = spec.shape((-1,4096)) # reshapes data into stacks of 4096 points
spec.shape # get the shape of the new array. The first number is the time integration

########################################
## save the text file to open an excell
########################################

np.savetxt("ReshapedData.csv",np.transpose(spec), delimiter=',') # saved to file ReshapedData.txt with a comma delimiter

########################################
## Plot
########################################

plot(spec[0]) # Plots the first integration
plot(spec[i]) # Plots the i-th intergration
```