Automatic Change of SDR Parameters' Values During Runtime in GNURADIO for Satellite Communication Subsystem

SSC21-S1-49

Automatic Change of SDR Parameters' Values During Runtime in GNURADIO for Satellite Communication Subsystem

Yasir M. O. ABBAS, Kenichi Asami
Kyushu Institute of Technology, Japan
1-1 Sensuicho, Kitakyushu, Fukuoka, Japan;
abbas.yasir-mohamed899@mail.kyutech.jp
asami@mns.kyutech.ac.jp

ABSTRACT

Shifting from traditional hardware radios to the Software-Defined Radio (SDR) is becoming reality, and SDRs are going rapidly to dominate the satellite communication subsystems. For testing designs, researchers use many tools such as the popular GNURADIO software which programs and controls SDR devices by providing signal processing blocks implementing the desired signals as well as hardware interface blocks. It is user-friendly and simple to beginners. Moreover, it has powerful and advanced capabilities for more complex missions.
In some cases, we need to modify communication parameters such as frequency, data rate or modulation scheme without relaunching the program. Many times, the values of these changes are not available until the runtime, thus parameter's new values need to be fed to the communication program while it is running.
As a case study, this paper presents a method of changing SDR transmit and receive frequency in GNURADIO to compensate for the doppler shift effect.
The main code that is generated by GNURADIO in Python is modified and linked with another Python program to calculate doppler shift frequencies. The real-time frequency value is fed to the SDR device blocks in GNURADIO while it is running using networking protocols. The frequency calculation code is based on PyEphem library. This program uses the two-line elements set (TLE) to know the satellite position then it uses the ground station coordinates as an input in order to find the relative velocity which is the main factor to calculate the doppler shift frequencies. The system is tested using a laptop, Raspberry Pi 4, LimeSDR and RTL-SDR devices.
Methods of handling such issues directly affect the efficiency of the communication which lead to more robust links to improve satellites data delivery capacity.
Keywords: SDR, GNURADIO, PyEphem, Doppler Shift, Satellite, Communication.

INTRODUCTION:

The system is labeled as reconfigurable or adaptable if there are easy ways to introduce changes on its parameters. During runtime of satellites communications systems some parameters require to be changed to increase efficiency, security, or reliability of the mission. This paper is experimenting changing the communication frequency in order to compensate for the Doppler shift effect.
The same method would work for changing gain, power, and modulation of the system. It can as well used to export and import variable values across systems or within the system parts. it allows the calculation tasks to be implemented in a separate code then only the results are provided to where they are useful.

GNURadio software

Is a free and open-source software development toolkit that provides signal processing blocks to implement software radios [1]. It is widely used to control software-defined radio (SDR) devices.

Software-Defined Radio

SDR refers to wireless communication in which the transmitter modulation is generated or defined by a computer. The receiver then also uses a computer to recover the signal intelligence [2]. In space-based system the computer is either SoC device or a field-programmable gate array (FPGA).
The most useful feature of the SDR is the ability to change radio parameters only by changing the software controlling the SDR. This is the
Many recent research is carried out to maximize the benefits SDR in reconfigurable and in adaptable radios.

Doppler Shift Effect

In communication systems in which receiver and transmitter are not fixed and have significant relative velocity, the transmitted signal is received with a frequency drift. This drift is proportional to the relative velocity between the two ends of the communication system.

XML-RCP protocol:
It is a remote procedure call (RPC) protocol, which uses XML to encode its calls. It uses Hypertext Transfer Protocol HTTP as its transport mechanism.

METHODOLOGY:

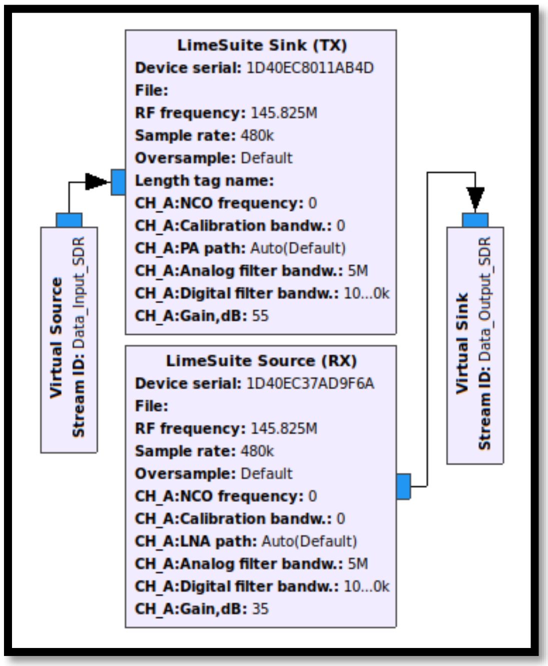
In Raspberry Pi 4, GNURadio software is used to control a LimeSDR mini device. A separate Python code (i.e., doppler frequency code) is written to control the frequency of the SDR.
This "doppler frequency code" is used to calculate the frequency of a satellite as received by an observer's ground station. It is developed in Windows 10 laptop by Python-3.9 using SpyderIDE in Anaconda environment. It uses PyEphem, which is a library that implements astronomical algorithms for Python programming language. It is free under the LGPL. The library is written in Python and C. After verifying the code in Windows, it is then implemented in a Raspberry Pi 4 device. Verifying the "doppler frequency code" is achieved by comparison with the resulting values of two well-known simulation software programs. i.e., Orbitron and HamRadioDeluxe running in windows.
The whole system verification is done by transmitting AX.25 packets between two LimeSDR mini devices while the frequency is being changed.

DESIGN AND IMPLEMENTATION:

Several approaches can be used for changing GNURadio variables. The best approach in terms of the simplicity and the ability to change the variables during runtime is found to use XML-PRC protocol.

Control SDR via GNURadio Flowgraph

GNURadio flowgraph is the main design platform for the code that sends and receives data. The flowgraph design and implementation will be explained in this session.
As shown in Figure 1, LimeSuite Sink (TX) block is the operational block to control the transmitted signals, the SDR device is selected by its serial number in the block properties. The frequency and the gain are defined as variables as in Figure 2. These variables' names should be noted because it will be used when scripting the code of the XML-RPC Clients.
Similarly, the received signal is captured from the SDR device by LimeSuite Source (RX) block.
The Virtual Source and Virtual Sink blocks in Figure 1 are the way of directing data flow to and from the LimeSDR devices. Data comes from a modulator to be transmitted by the transmitter SDR. The receiving SDR captures the signal and forward it to the designed demodulator in GNURadio. Audio frequency shift keying (AFSK) modulation is used to exchange AX.25 packets, the design of the modulator and the demodulator themselves are out of the scope of this paper, any other modulation can be used.

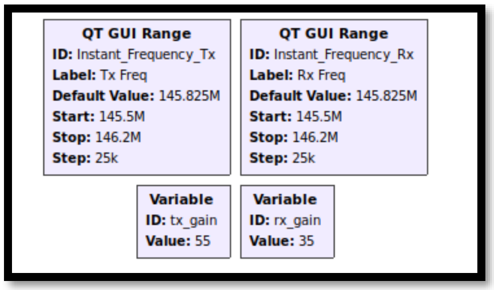

LimeSuite Sink (TX)
Device serial: 1D40EC8011AB4D
File:
RF frequency: 145.825M
Sample rate: 480k
Oversample: Default
Length tag name:
CH_A:NCO frequency: 0
CH_A:Calibration bandw.: 0
CH_A:PA path: Auto(Default)
CH_A:Analog filter bandw.: 5M
CH_A:Digital filter bandw.: 10...0k
CH_A:Gain,dB: 55

LimeSuite Source (RX)
Device serial: 1D40EC37AD9F6A
File:
RF frequency: 145.825M
Sample rate: 480k
Oversample: Default
CH_A:NCO frequency: 0
CH_A:Calibration bandw.: 0
CH_A:LNA path: Auto(Default)
CH_A:Analog filter bandw.: 5M
CH_A:Digital filter bandw.: 10...0k
CH_A:Gain,dB: 35

Virtual Source
Stream ID: Data_Input_SDR

Virtual Sink
Stream ID: Data_Output_SDR

Figure 1.


QT GUI Range
ID: Instant_Frequency_Tx
Label: Tx Freq
Default Value: 145.825M
Start: 145.5M
Stop: 146.2M
Step: 25k

QT GUI Range
ID: Instant_Frequency_Rx
Label: Rx Freq
Default Value: 145.825M
Start: 145.5M
Stop: 146.2M
Step: 25k

Variable
ID: tx_gain
Value: 55

Variable
ID: rx_gain
Value: 35

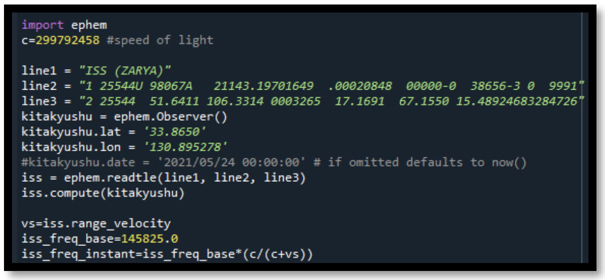
Figure 2.

The Calculation of the doppler frequency

This Python code imports "ephem" package. Then it gets the position information from the Two-Line-Element (TLE) format. The location of the observer's ground station and the base-frequency of the satellite should be provided.
The code calculates the range velocity, which is the relative velocity of the satellite with respect to the ground station. From this value the code calculates the doppler shift frequency using Equation 1 and 2 as shown in Figure 3.

$$Fi = Fb + Fd \quad \ldots\ldots\ldots\ldots.. (1)$$
$$Fd = Fb*C/(C+Vs) \quad \ldots\ldots. (2)$$

Where:
Fi: Instant Frequency.
Fb: Base Frequency.
Fd: Doppler Shift Frequency.
C: Speed of light.
Vs: Range Velocity.



```
import ephem
c=299792458 #speed of light

line1 = "ISS (ZARYA)"
line2 = "1 25544U 98067A   21143.19701649  .00020848  00000-0  38656-3 0  9991"
line3 = "2 25544  51.6411 106.3314 0003265  17.1691  67.1550 15.48924683284726"
kitakyushu = ephem.Observer()
kitakyushu.lat = '33.8650'
kitakyushu.lon = '130.895278'
#kitakyushu.date = '2021/05/24 00:00:00' # if omitted defaults to now()
iss = ephem.readtle(line1, line2, line3)
iss.compute(kitakyushu)

vs=iss.range_velocity
iss_freq_base=145825.0
iss_freq_instant=iss_freq_base*(c/(c+vs))
```

Figure 3.

XML-RPC Protocol implementation

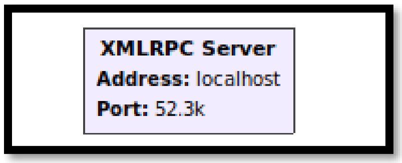
The XML-RPC Protocol needs a server to be launched in the controlled side and a client in the controlling side.

XML-RPC Server Implementation

The server opens a gate to remotely access GNURadio functions that sets the values on the flowgraph variables. We will use it to control the earlier explained SDR variables in the GNURadio.

For implementation, GNURadio XMLRPC Server block is used as shown in Figure 4.

As the operation code and the control code are both in the same device, we use localhost IP address "127.0. 0.1". The port number is selected by the developer. It is chosen to be 52300.



XMLRPC Server
Address: localhost
Port: 52.3k

Figure 4.

XML-RPC Client implementation:

The XML-RPC client is used as the controller. It is where the input is entered. It is implemented using Python code that executes the callback on the server when a variable need to be changed. Figure 5 is showing the function that changes SDR parameters once. It gets the values from global variables. This function loops within the ground station when tracking a satellite, the frequency is changed every loop according to the values calculated by doppler frequency code.

```
def changeVariables():
  global Instant_Frequency_Tx
  global Instant_Frequency_Rx
  global tx_gain
  global rx_gain
  xmlrpcDopplerFreq = xmlrpclib.Server('http://localhost:52300')
  xmlrpcDopplerFreq.set_Instant_Frequency_Tx( Instant_Frequency_Tx )
  xmlrpcDopplerFreq.set_Instant_Frequency_Rx( Instant_Frequency_Rx )
  xmlrpcDopplerFreq.set_tx_gain( tx_gain )
  xmlrpcDopplerFreq.set_rx_gain( rx_gain )
```

Figure 5

Note that the naming of "set_Instant_Frequency_Tx, set_Instant_Frequency_Tx, set_tx_gain and set_rx_gain" is based on the code generated by the GNURadio flowgraph.

TESTS, RESULTS AND VALIDATION:

Two tests are performed to validate the system. One is to validate the doppler frequency. The second test is to ensure that the XML-RPC protocol is able to control the SDR.

The test setup -as in Figure 6- consists of two LimeSDR mini devices connected to laptop running Ubuntu Operating System. Using GNURadio software data is sent between the SDR devices.

The external Python code to control the frequency values of a simulated satellite is integrated with the control code. The satellite TLE is chosen to be in the ISS orbit and the base frequency is set to 145.825 MHz. The ground station is assumed to be in Kitakyushu, Japan.

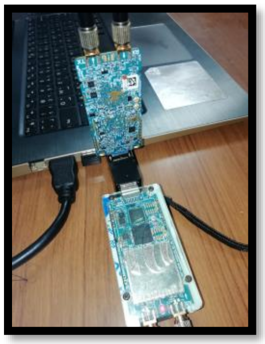
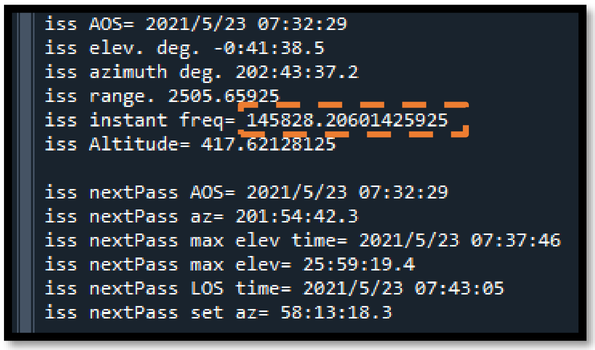


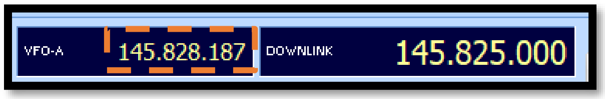
Figure 6.

The doppler frequency code, which generates the instant frequency from the TLE, is tested with the international space station (ISS). The results are compared with the outputs of the Orbitron software program and HamRadioDeluxe program as shown in Figures 7, 8, 9 and 10.



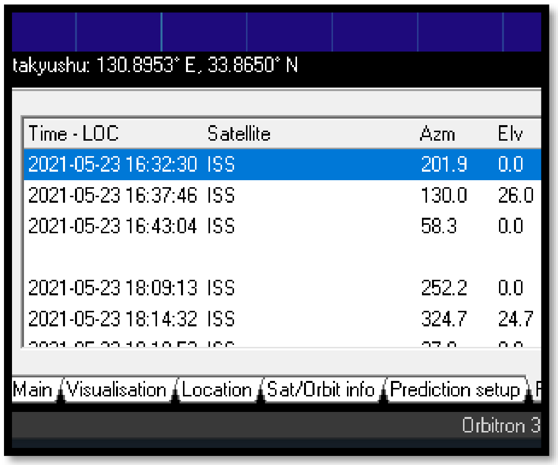
```
iss AOS= 2021/5/23 07:32:29
iss elev. deg. -0:41:38.5
iss azimuth deg. 202:43:37.2
iss range. 2505.65925
iss instant freq= 145828.20601425925
iss Altitude= 417.62128125

iss nextPass AOS= 2021/5/23 07:32:29
iss nextPass az= 201:54:42.3
iss nextPass max elev time= 2021/5/23 07:37:46
iss nextPass max elev= 25:59:19.4
iss nextPass LOS time= 2021/5/23 07:43:05
iss nextPass set az= 58:13:18.3
```

Figure 7.



VFO-A 145.828.187 DOWNLINK 145.825.000

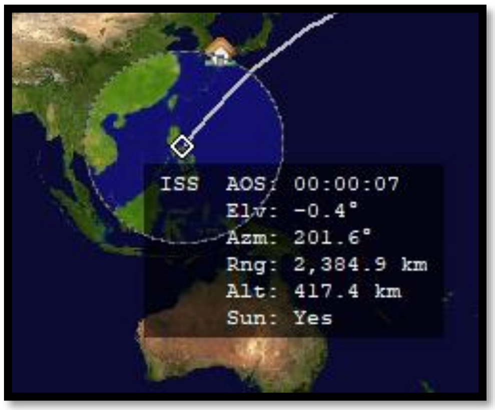
Figure 8.



takyushu: 130.8953° E, 33.8650° N

Time - LOC	Satellite	Azm	Elv
2021-05-23 16:32:30	ISS	201.9	0.0
2021-05-23 16:37:46	ISS	130.0	26.0
2021-05-23 16:43:04	ISS	58.3	0.0
2021-05-23 18:09:13	ISS	252.2	0.0
2021-05-23 18:14:32	ISS	324.7	24.7

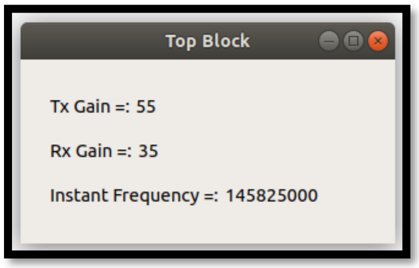
Figure 9.



```
ISS  AOS: 00:00:07
     Elv: -0.4°
     Azm: 201.6°
     Rng: 2,384.9 km
     Alt: 417.4 km
     Sun: Yes
```

Figure 10.

To test the controlling code the frequencies are monitored using "QT GUI Label" block of the GNURadio. The calculated value that is sent via XML-RPC protocol should appears as in Figure 11. It should be changing because the satellite position is changing with time.



Top Block

Tx Gain =: 55

Rx Gain =: 35

Instant Frequency =: 145825000

Figure 11

The reception of the transmitted data is confirmed listening to the AFSK packet sound and by graphs shown in Figure 12.

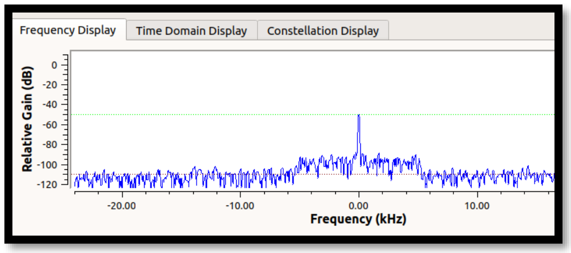


Figure 12-a: Transmitted signal at the receiving SDR

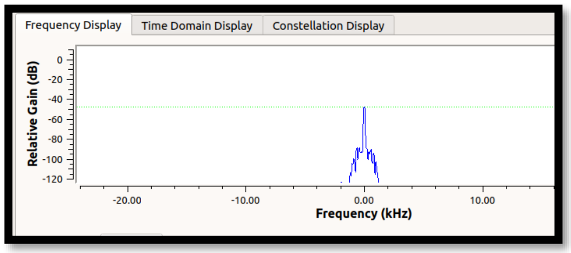


Figure 12-b: The received signal after filtering.

DISCUSSION:

The measured deviation in time is found less than 1 second. For the frequency, deviation it is found less than 30Hz. This precision is within the acceptable range for the Ground Station satellite communication. Comparing Figure 7 and Figure 9, the time is 9-hours different because of the time zones. For better validation, the system needs to be tested with a real satellite because here in this setup both SDRs are fixed in place and the frequency is changed in both to simulate the moving effect.

CONCLUSION

This paper shows the details of using XML-RPC protocol to control SDR parameters. This is handy to implement a tracking ground station or a satellite hopping communication system. It can also be used in adaptive systems to control the noise and the flowchart parameters during runtime [3] and in many other tasks.

REFERENCES

1. GNURadio, https://www.gnuradio.org/ (accessed on 1-June-2021)
2. M. N. O. Sadiku and C. M. Akujuobi, "Software-defined radio: a brief overview," in IEEE Potentials, vol. 23, no. 4, pp. 14-15, Oct.-Nov. 2004, doi: 10.1109/MP.2004.1343223.
3. ABBAS, Yasir M.O.; Asami, Kenichi. 2021. "Design of Software-Defined Radio-Based Adaptable Packet Communication System for Small Satellites" Aerospace 8, no. 6: 159. https://doi.org/10.3390/aerospace8060159.