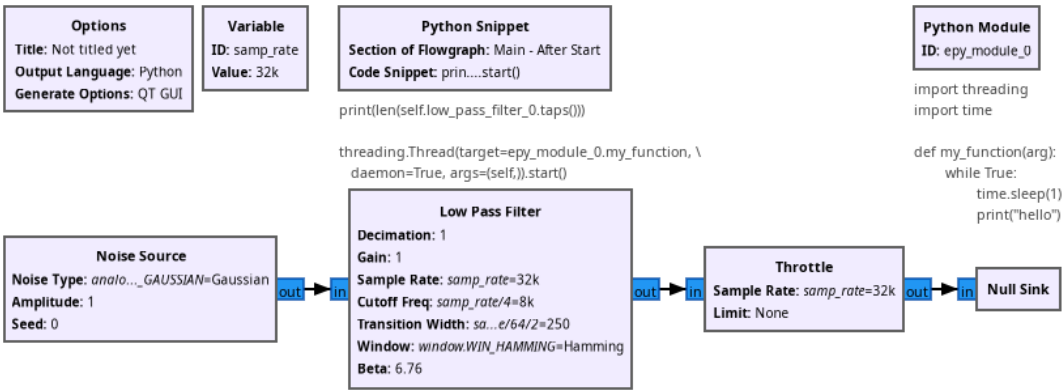
Introduction

1. Tutorial
   - getting familiar with GNU Radio Companion (3.10)
   - signal types, variables (static/dynamic through UI),
   - basics of discrete time signal processing (real v.s complex, aliasing),
   - time and frequency domain representations,
   - filter characteristics (taps v.s transition bandwidth)
   - characterization of the low pass filter with sweeping sine wave and with noise source
   - introduction to the Python generated by GNU Radio Companion, display of the tap length as a function of filter transition width
2. Tutorial
   - gaussian noise generator, histogram display
   - Python block for gaussian fit of the histogram
   - Repeat with real signal collected at GSI, file Source and SigMF meta data format, reference to IQengine.org
3. Tutorial: communicating with external tools (`https://github.com/jmfriedt/gnuradio_communication` and `https://gitlab.com/gnuradio_book/flowcharts`, Chap. 3)
   - named pipes
   - Zero-MQ
   - XML RPC
   - practical example with correlations (GNU Radio 3.10) on synthetic data

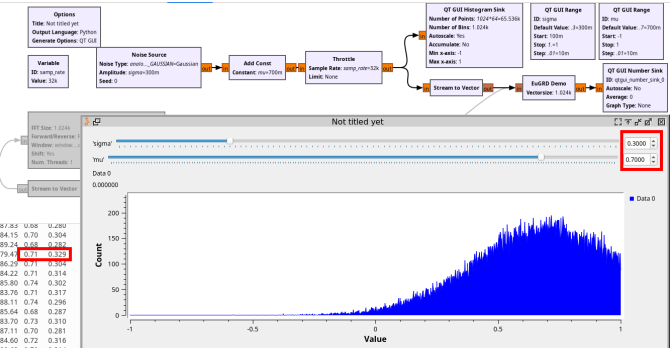
Python snippet & Python module

- GNU Radio Companion is a Python code generator...
- ... so we can mix custom Python code with auto-generated code.
- If we modify generated code, no way to return to the graphical interface.
- Solutions: `Python Snippet` and `Python Module`



Option `daemon=True` when spawning the thread: terminates with end of the flowgraph execution.
These Python snippet and module **cannot** access IQ data, only control dataflow

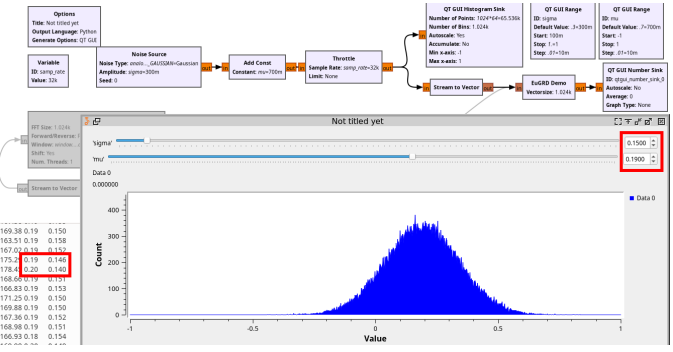
Processing IQ stream with a custom Python block




```python
import numpy as np
from gnuradio import gr
from scipy.optimize import curve_fit
# https://github.com/zaratustrah/curve-fitting/blob/main/curve_fit.py

class blk(gr.decim_block):
  def __init__(self, vectorSize=1024):
    gr.sync_block.__init__(
      self,
      name='EuGRD Demo',   # will show up in GRC
      in_sig=[(np.float32,vectorSize)],
      out_sig=[np.float32]
    )
  def fit_function(self, x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2.*sigma**2))
  def work(self, input_items, output_items):
    for k in range(0,len(output_items)) :
      y,x=np.histogram(input_items[0][k],bins=32,range=(-1,1))
      x=0.5*(x[0:-1]+x[1:])  # x=middle of each bin
# p = [1., 1., .1]  initial conditions could be estimated
      popt, pcov = curve_fit(self.fit_function, x, y) # , p0=p
      print(f"{popt[0]:.2f}\t{popt[1]:.2f}\t{np.abs(popt[2])
          :.3f}")
# sigma ** 2 so either positive or negative solutions are
      output_items[0][k]=np.abs(popt[2])
    return len(output_items[0])
```

Processing IQ stream with a custom Python block

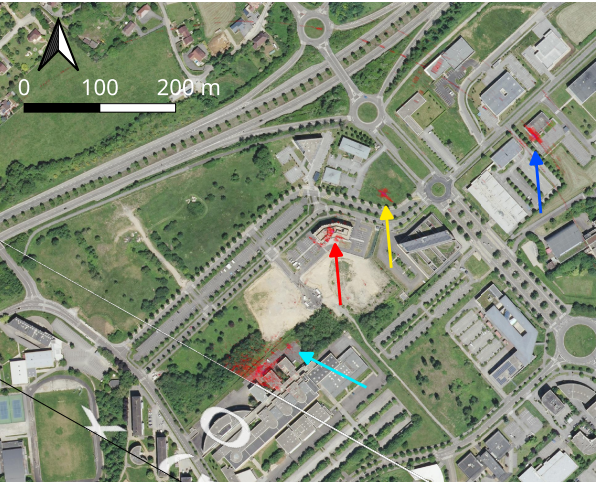



```
169.38 0.19  0.150
163.51 0.19  0.158
167.02 0.19  0.152
175.2  0.19  0.146
178.4  0.20  0.140
168.66 0.19  0.151
166.83 0.19  0.153
171.25 0.19  0.150
168.98 0.19  0.150
167.36 0.19  0.152
168.98 0.19  0.151
166.93 0.18  0.154
```

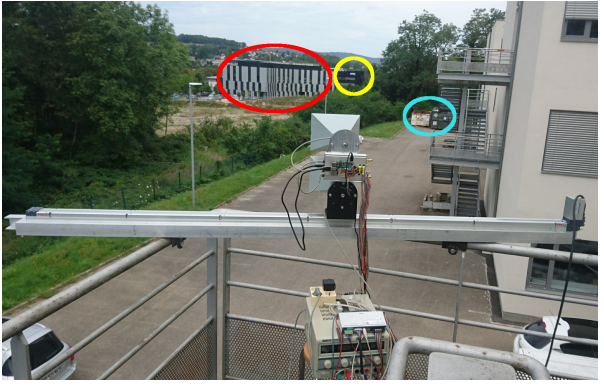
```python
import numpy as np
from gnuradio import gr
from scipy.optimize import curve_fit
# https://github.com/zaratustrah/curve-fitting/blob/main/curve_fit.py

class blk(gr.decim_block):
  def __init__(self, vectorSize=1024):
    gr.sync_block.__init__(
      self,
      name='EuGRD Demo',    # will show up in GRC
      in_sig=[(np.float32,vectorSize)],
      out_sig=[np.float32]
    )
  def fit_function(self, x, A, mu, sigma):
    return A*np.exp(-(x-mu)**2/(2.*sigma**2))
  def work(self, input_items, output_items):
    for k in range(0,len(output_items)) :
      y,x=np.histogram(input_items[0][k],bins=32,range=(-1,1))
      x=0.5*(x[0:-1]+x[1:])  # x=middle of each bin
# p = [1., 1., .1]  initial conditions could be estimated
      popt, pcov = curve_fit(self.fit_function, x, y) # , p0=p
      print(f"{popt[0]:.2f}\t{popt[1]:.2f}\t{np.abs(popt[2])
          :.3f}")
# sigma ** 2 so either positive or negative solutions are
      output_items[0][k]=np.abs(popt[2])
    return len(output_items[0])
```

Ground-Based Synthetic Aperture RADAR (SDR-GB-SAR)[1]



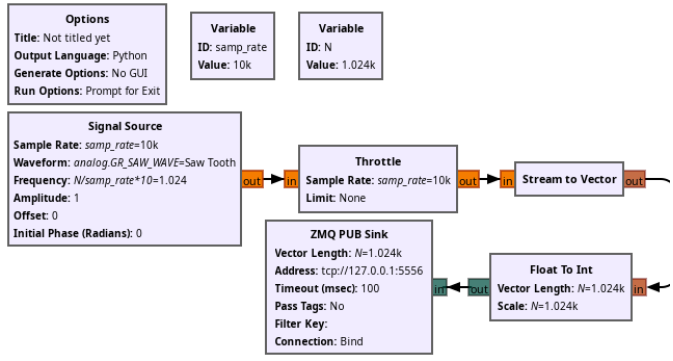
- Some trees to the left blocking the view, but antenna beam is only 18° wide anyway (20 dBi antenna)
- measurement at > 500 m range
- nearby metallic container (cyan), newly built building not yet visible on Google Maps (yellow)



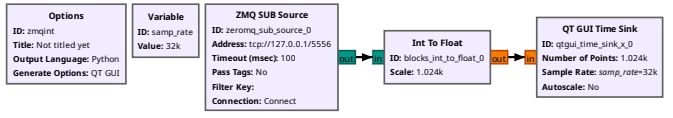
[1] https://fosdem.org/2024/schedule/event/fosdem-2024-2050-covert-ground-based-synthetic-aperture-radar-using-a-wifi-emitter-and-sdr-receiver/

ZeroMQ Publish-Subscribe

TX:



RX:



Python:

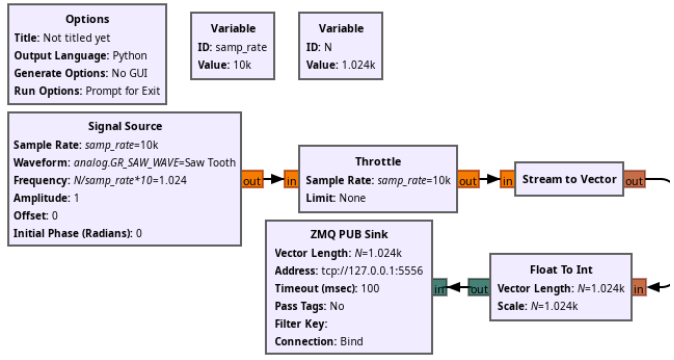
```python
import numpy as np # pkg load signal;
import zmq         # pkg load zeromq;
import array
from matplotlib import pyplot as plt

Nt=256
context=zmq.Context()

sock1=context.socket(zmq.SUB) # sock1=zmq_socket(ZMQ_SUB);
sock1.connect("tcp://127.0.0.1:5556");
sock1.setsockopt(zmq.SUBSCRIBE, b"")
vector1=[]
while (len(vector1)<Nt):
  raw_recv=sock1.recv()
#  recv=array.array('f',raw_recv)
  recv=array.array('i',raw_recv)
  print(f"{recv[0]} {recv[1]} {recv[-1]}")
  plt.plot(recv)
  plt.show()
#    vector1tmp=recv[0::2]
#    vector2tmp=recv[1::2]
```

ZeroMQ Publish-Subscribe

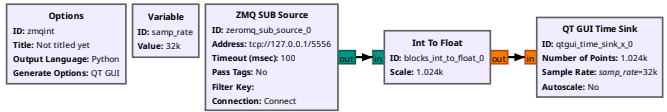
TX:



GNU/Octave:

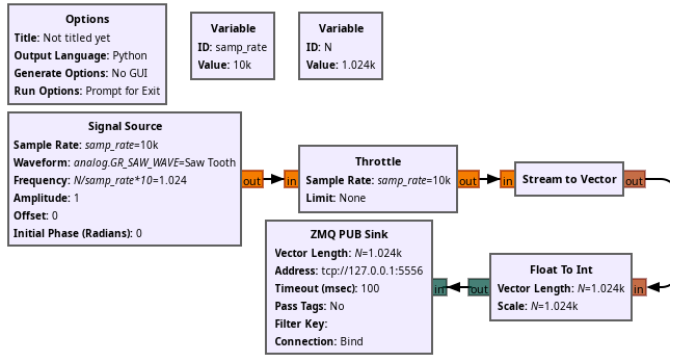
```
pkg load zeromq;
Nt=1024
sock1=zmq_socket(ZMQ_SUB);
zmq_connect(sock1,"tcp://127.0.0.1:5556");
zmq_setsockopt(sock1, ZMQ_SUBSCRIBE, "");
recv=zmq_recv(sock1, Nt*8, 0);
% vector=typecast(recv,"single complex");
vector=typecast(recv,"int32");
vector(1), vector(2),vector(end)
plot(vector)
pause
```

RX:

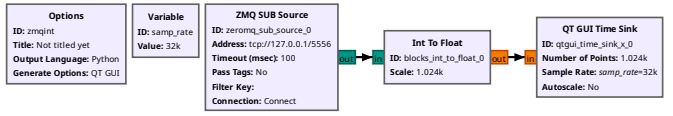


ZeroMQ Publish-Subscribe

TX:



RX:



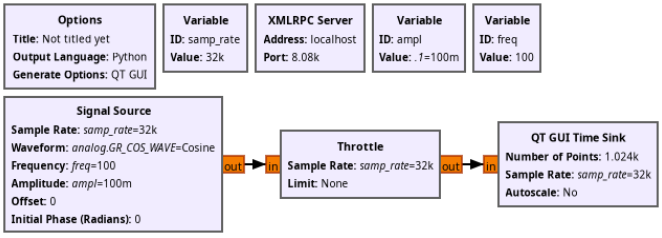
C:

```c
#include <stdio.h>
#include <zmq.h>
#include <stdint.h>

int main()
{   int32_t *res;
    void *context = zmq_ctx_new();
    void *subscriber = zmq_socket(context, ZMQ_SUB);
    char message[1024*4];
    int len;
    zmq_connect(subscriber, "tcp://127.0.0.1:5556");
    zmq_setsockopt(subscriber, ZMQ_SUBSCRIBE, "", 0);
    res=(int32_t*)(message);
    while (1) {
        len=zmq_recv(subscriber, message, 1024*4, 0);
        if (len!=-1) {printf("%d: %ld %ld %ld\n",len,(res[0]),r
            ↪[1023]));}
        else printf("error\n");
    }
    zmq_close(subscriber);
    zmq_ctx_destroy(context);
    return 0;
}
```

XMLRPC for flowgraph control

TX:



Bash:

```
xmlrpc localhost:8080 set_freq i/200
```

Python:

```
import xmlrpc.client # Remote Procedure Call method that →
      ↪uses XML passed via HTTP(S) as a transport
proxy = xmlrpc.client.ServerProxy("http://localhost:8080/")

for method_name in proxy.system.listMethods():
    if (method_name.find("set_")>=0):
        print(method_name)

try:
    setampl=proxy.set_ampl(0.2)
except xmlrpc.client.Fault as err:
    print("Unsupported function")
try:
    setfreq=proxy.set_freq(200)
except xmlrpc.client.Fault as err:
    print("Unsupported function")
```

Correlation

How to accurately find a time delayed copy of a signal in noise? Cross correlation:

$$xcorr(p, s)(\tau) = \int s(t) \cdot p^*(t + \tau) dt$$

RADAR range resolution: $\Delta R = \frac{c}{2B}$ with c the speed of light

Many ways of spreading the spectrum to reach B:

- short pulse of duration dt ($B \simeq 1/dt$) – pulsed RADAR
- frequency sweep from f_{start} to f_{stop}: $B = f_{stop} - f_{start}$ – FSCW
- beatnote between outgoing and backscattered linearly swept radiofrequency wave – FMCW
- pseudo random sequence – noise RADAR

Cross correlation peak width is $1/B$, and SNR improvement is

$$\underbrace{PCR}_{PulseCompressionRatio} = \underbrace{B}_{bandwidth} \times \underbrace{T}_{duration} = N$$

number of symbols in digital version

However, the cross-correlation block **does not** exist in GNU Radio...

Spectrum spreading numerical experiments

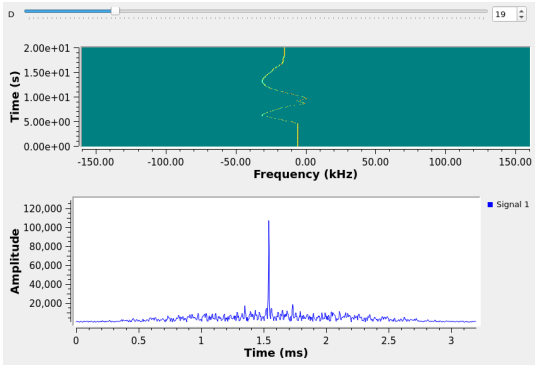
From convolution to correlation:

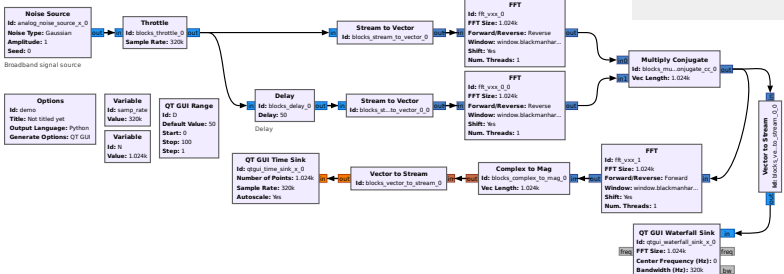
- Convolution: $conv(s,r)(\tau) = \int s(t)r(\tau - t)dt$
- Practical computation of convolution:

$$FT(conv(s,r)) = FT(s) \cdot FT(r)$$

- Correlation: $corr(s,r)(\tau) = \int s(t)r^*(t+\tau)dt$
- Convolution $\rightarrow$ correlation: time reversal
- since $\exp(j\omega t)^* = \exp(-j\omega t)$, we conclude

$$FT(corr(s,r)) = FT(s) \cdot FT^*(r)$$





Spectrum spreading numerical experiments

From convolution to correlation:

- Convolution: $conv(s,r)(\tau) = \int s(t)r(\tau - t)dt$
- Practical computation of convolution:

$$FT(conv(s,r)) = FT(s) \cdot FT(r)$$

- Convolution / correlation: t[illegible]

$$FT(corr(s,r)) = FT(s) \cdot FT^*(r)$$