GNURadio

THE FREE & OPEN SOFTWARE RADIO ECOSYSTEM

1 Installation

gnuradio–install.sh

```
# Install pybombs via python-pip
sudo pip install PyBOMBS

# Add recipe lists from git repositories
pybombs recipes add gr-recipes git+https://
    ↪ github.com/gnuradio/gr-recipes.git
pybombs recipes add gr-etcetera git+https://
    ↪ github.com/gnuradio/gr-etcetera.git

# Set installation folder to '~/Desktop/
    ↪ pybombs'
pybombs prefix init ~/Desktop/pybombs -a
    ↪ myprefix

# Enable documentation
pybombs config builddocs=ON

# Run gnuradio installation with verbose
    ↪ output
pybombs -vv install gnuradio

# Publish install variables as environment
    ↪ variables
source ~/Desktop/pybombs/setup_env.sh

# Apply also after re-booting
echo 'source ~/Desktop/pybombs/setup_env.sh'
    ↪  >> ~/.profile
echo 'source ~/Desktop/pybombs/setup_env.sh'
    ↪  >> ~/.bashrc

# Run GNU Radio Companion
gnuradio-companion
```

2 Getting Started

getting–started.py

```
from gnuradio import gr, blocks, filter,
    ↪ analog

class my_topblock(gr.top_block):
    def __init__(self):
        gr.top_block.__init__(self)
        amp = 1
        taps = filter.firdes.low_pass(1, 1,
    ↪   0.1, 0.01)
        self.src = analog.noise_source_c(
    ↪   analog.GR_GAUSSIAN, amp)
        self.flt = filter.fir_filter_ccf(1,
    ↪   taps)
        self.snk = blocks.null_sink(gr.
    ↪   sizeof_gr_complex)
        self.connect(self.src, self.flt,
    ↪   self.snk)

if __name__ == "__main__":
    tb = my_topblock()
    tb.start()
    tb.wait()
```

3 Gnu Radio Basics

3.1 Create Hierarchical Block

inputLayer.py

```
from gnuradio import blocks
from gnuradio import fft
from gnuradio import gr
from gnuradio.fft import window

class inputLayer(gr.hier_block2):
    def __init__(self, vlen=64):
        gr.hier_block2.__init__(
            self, "Input Layer",
            gr.io_signature(
                1, 1,
                gr.sizeof_gr_complex*vlen
            ),
            gr.io_signature(
                1, 1,
                gr.sizeof_float*vlen
            ),
        )

        # Blocks
        fft = fft.fft_vcc(
            vlen, True,
            (window.rectangular(vlen)),
            True, 8
        )
        log = blocks.nlog10_ff(20, vlen, 0)
        mag = blocks.complex_to_mag(vlen)
        mult = blocks.multiply_const_vcc(
            ([1./float(vlen), ]*vlen)
        )

        # Connections
        self.connect(self, mult)
        self.connect(mult, fft)
        self.connect(fft, mag)
        self.connect(mag, log)
        self.connect(log, self)
```

3.2 Create Python Block

vector_sum_vff.py

```
import numpy
from gnuradio import gr

class vector_sum_vff(gr.sync_block):
    def __init__(self, vlen):
        self.vlen = vlen
        gr.sync_block.__init__(self,
            name="vector_sum_vff",
            in_sig=[(numpy.float32, vlen)],
            out_sig=[(numpy.float32, 1)])

    def work(self,input_items,output_items):
        in0 = input_items[0]
        out = output_items[0]
        out[:] = numpy.sum(in0[0:1], axis=1)
        return 1
```

3.3 Create C++ Block

3.4 Streams and Vectors

3.5 Stream Tags

3.6 Message Passing

3.7 Performance Monitoring

4 Signal Processing

4.1 Channel Models

4.2 Digital Modulation

4.3 Filtering

4.4 Resampling

4.5 FFT