UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

DRONE DETECTION AND INHIBITION

A Master's Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Erick Medina Moreno

In partial fulfilment

of the requirements for the degree of

MASTER IN TELECOMMUNICATIONS ENGINEERING

Advisor: Josep Paradells

Barcelona, February 2020

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

telecom BCN

Title of the thesis: **Drone Detection and Inhibition**

Author: **Erick Medina Moreno**

Advisor: **Josep Paradells**

Abstract

In the recent years, there has been an increase in the demand and use of unmanned aerial vehicles (UAV) know as drones. Even though there are restrictions for its use by licensed drivers and in determined areas, it has been notorious that these devices are used carelessly in open areas putting in risk the integrity of people. In order to tackle this issues, we propose a compound device composed of a Software Defined Radio (SDR) and a processing unit that can help detect drones and interfere its communications with its controller. We have built a tool that can identify unusual signals that can help an operator to detect the presence of drones. Our results have shown that we can help detect drones in the 80% of the cases.

TODO (continue)

Dedication: TODO

Acknowledgements

Gratitude to Ecuador's “Secretaría de Educación Superior, Ciencia y Tecnología” (Senescyt) as the scolarship's sponsor.

Teachers, mentors, etc.

Revision history and approval record

Revision	Date	Purpose
0	09/12/2019	Document creation
1	dd/mm/yyyy	Document revision

Written by:		Reviewed and approved by:	
Date	09/12/2019	Date	dd/mm/yyyy
Name	Erick Medina	Name	Zzzzzzz Wwwwwww
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables

Each table in the thesis must be listed in the “List of Tables” and each must be given a page number for easy location.

List of Code Snippets

Keywords-Acronyms:

SDR: Software Defined Radio

RF: Radio frequency

FFT: Fast Fourier Transform

OS: Operative System

UAV: Unmanned aerial vehicles

1. Introduction

The proliferation in the demand and use of unmanned aerial vehicles (UAV) otherwise know as drones, due to its affordable cost and its multimedia capabilities have posed a threat to security, since they can be used by unlicensed drivers and in areas that are not authorised for its flight, such as natural protected areas or touristic points of interest.

To tackle this problem, we must follow two lines of work in order to take actions to countermeasure the drone presence. The first one is the drone detection, that can be handled with different alternatives such as sound recognition, image recognition, radar detection and radio detection. The other one is the drone control or takedown which can include radio interference, radio control, laser guns attacks, or physical catch.

In the scope of this project we will be using both radio detection and radio interference as the means to detect and takedown the drone communications.

To accomplish this task we will use a Software Defined Radio (SDR) peripheral called HackRF One, and the GNU Radio software development kit, tools that will be introduced in detail in the following chapters. The programming language Python will be used along with GNU Radio in order to create custom components according to the needs of this project.

Also some models of drones from different manufacturers will be used, for which we will study its communications with the remote controller, capture the signals so we could use them in the identification process, and verify that we can interfere the drone communications.

1.1. Objectives

- Study the properties and functions of the HackRF and identify the characteristics that suite best for our project.
- Setup the GNU Radio development environment in a Linux machine.
- Study the GNU Radio Companion software, its processing blocks and mechanisms to create custom blocks.
- Study drone communications.
- Design a software that can help in the drone detection and it's signal interference.
- Test the implementation of the drone detection and interference mechanisms.

2. State of the art of the technology used or applied in this thesis

A background and a comprehensive review of the literature is required. This is known as the Review of Literature and should include relevant recent research conducted on the subject matter.

2.1. Unmanned Aerial Vehicles (UAV) or Drones

Unmanned aerial vehicles as it names states are vehicles that can fly without the need of a human pilot on board. Commercially they are also know as drones. They can be used for different purposes going from security and military, up to multimedia acquisition, and their applications keep expanding.

Certainly in the last decade, it has been their capacity to obtain aerial photographies and videos, the main cause of their soaring popularity. And since the materials and electronic components to build commercial drones have plummeted, their acquisition price has made it become an affordable gadget to a broader sector of the population, not only to aeronautics enthusiasts.

Drones can be found freely available to buy on the internet and physically in retailers and toy stores. Even though in some countries like Spain, you need a license to operate drones, it is not a requirement to buy them, making it possible that unqualified people have access to controlling these types of aircraft.

Aeronautical regulations in Spain state the requirement of a license to pilot remotely a drone professionally, they also indicate that in areas with people and buildings it is not allowed to fly a drone unless authorised explicitly.

Even regulation exist, there have been a few security-incidents related to drone activity. The most recent, as of the date of writing of this document, was reported in Madrid in February 2020, with the shutdown of the aerial space surrounding the Barajas airport for more than two hours affecting 26 flights. All of this due to the unauthorised presence of drones near the airport, that were detected visually by pilots.

Another recent cases occurred in Barcelona between October and December 2019, with a failed display of banners carried by drones inside the Camp Nou Stadium in a Barcelona-Real Madrid match, and the detection of around 83 unauthorised drones in the streets of Barcelona during independence related protests.

Indeed drones, pose a security threat for people and governments if they are not effectively controlled. It is crucial to understand how drones work in order to choose the right strategies to detect and disable them safely.

2.2. How drones work? An introduction

Drones are composed of both hardware and software parts within its body. The hardware components are in charge of providing the aircraft the ability to takeoff, land, fly,manoeuvre, communicate and sense. Software is in charge of controlling the aircraft.

Propellers are in charge of generating torque and thrust for all movement related actions. Depending on the drone characteristic the manufacturer chan choose the quantity, number of blades, length and material of propellers, but one of the most widely known design is the four propeller drone in the form of a quadcopter.

[Quadcopter figure]

Motors are the components in charge of providing the torque and thrust to the propellers. Most common models are the brushless motors, and the most important characteristics should be chosen according to the drone's total weight, and they are efficiency, revolutions per minute, and thrust.

The electronic speed controller (ESC) is in charge of providing the voltage to the motors, and change the drone's speed and direction according to the voltage provided, among other features. This component is fundamental for radio controlled drones.

Battery is in charge of providing the energy to the drone and mainly to the motors. Although its characteristics depend on the drone's features, one of the most commonly used is the Lithium Polymer (LiPo) rechargeable battery.

Sensors included within a drone can vary according to its features, but most frequently we can find distance sensors and orientation sensors. They are in charge of feeding the flight controller with information such as accelerations, movement changes, distance to closest objects. If the drone has radio capabilities, here we will find the radio sensors that allow communication with a remote controller or with other telecommunications providers.

The flight controller is in charge of controlling all the drone's functions. Here all the sensor's information is processed and taken into account to make flight decisions. If the drone is configured to work in standalone mode, all the flight instructions are set here. If it is configured in a remote controlled mode, all the information received from the radio controller is processed and passed into the motors in order to execute the user's manoeuvres.

The software of a drone is embedded in its flight controller. All decisions made by it are programmed in software, and take into account the data from the different sensors. The language used depends on the flight controller, but the most used are Pyhton, C, C++ or C#.

The remote controller is an external component of the drone. It allows to manoeuvre the drone. It provides buttons and rods that change the direction, angle and altitude. It has an antenna that emits the control data to the drone. Depending on the complexity of the drone it can also show telemetry data from the drone and multimedia.

[quote The Basics of Quadcopter Anatomy]

The previously described components are the most common found in commercial drones. But as it has been stated before, depending on the purpose of the drone, more complex components can be found in it.

There are drones equipped with water hoses that can help extinguish fires [quote DJI R&D] and xompanies such as Aerones and Walkera are working closely with firefighters department to make them a reality.

Farming drones are equipped with heat camera sensor, to control the cattle, but these have been used successfully in rescue scenarios. In October 2019 in Minnesota, United States of America, a farming drone helped find a missing child in a forest with the use of the heat camera.[quote Drone Kid missing]

2.3. Drone footprints and detection

Since the components of drones have been already explained in the previous chapter, we can start discussing about the traces or footprints that they leave while in use. We could classify them in visual, audible, and radio footprints.

Visually a drone can be identified when in the air by its shape, which is in most cases a quadcopter. It can become more complicated if a drone has airplane-like structure or more complex structures as in military uses. With a photo or video camera it is possible to develop drone image recognition algorithms. Also with the use of radars the presence of a drone can be sensed.

When the drone is flying, the motors and propellers moving the air produce a characteristic noise that leaves an audible footprint. The intensity of the noise can vary from model to model. With the use of microphones it can be possible to develop drone recognition algorithms.

If the drone uses radio communications with a remote controller while it is flying, then it leaves a radio-frequency footprint with all the data transmission between them. With the use of radio receivers it is possible to sense the presence of drone communications.

It can be discussed with a broader perspective the pros and cons of each one of the four drone detection possibilities mentioned in the above paragraphs. Nevertheless, we will focus on the radio-frequency detection approach, since for this project, radio receiver equipment has been provided for the task.

2.4. Drone communications

For the scope of this work, we will focus on drones that have the capability to communicate with a remote controller, and will use the radio-frequency footprint detection approach. And for this purpose it is fundamental to understand how communications work in a drone.

TODO

Remote Control for drones

PROTOCOLS

What information is transferred: telemetry, orders,

Differences between manufacturers

GPS autonomous flights

2.5. How to tackle unauthorized drones?

Police drones

Lasers

2.6. SDR and HackRf

Bits per sample

AS source

As Sink

2.7. GNU RADIO

2.8. Data acquisition and interpretation in GNU Radio with HackRf

FFT size

sample rate

2.9. Blocks in GNU Radio

Osmocom block

Custom blocks

3. Methodology / project development

For the development of the project we will use the following components:

HARDWARE

- Laptop Dell Inspiron, Intel Core i3 1.90GHz 64bit quadcore, 8GiB RAM, Linux Ubuntu 18.04.4 LTS.
- Raspberry Pi 3B+, 1.4GHz 64bit quadcore, 1GiB SD RAM, Linux Ubuntu Server for Raspberry 3.
- HackRF One
- DJI Mavic Pro and its remote controller
- SICE Why Evo universal remote controller

SOFTWARE

- Python 2.7
- GnuRadio framework and GNURadio Companion
- QT Framework

Even though the GnuRadio framework can be installed in MacOS and Windows, it was chosen to work in a Linux computer, due to an easier installation process and more stability. In a MacOS environment, an update of the OS or a specific program such as XCode can make the framework unusable.

The process consists of making the project in the laptop computer and then port it to the Raspberry Pi, being fundamental that both devices have the same versions of Python and GNURadio installed for a correct compatibility.

For the project we have programmed five executable Python scripts initially created in GNURadio Companion but later modified as wished according to our real needs. Additionally we have created three custom GNURadio components under the *gr-tfm* package which will be in charge of processing data and create a file database.

The main outcome of this project is a computer software that helps in the detection of drone RF activity and the inhibition of its signals. Python is the chosen programming language to develop this task, because of the compatibility with both GNURadio and QT frameworks. In fact, as we have mentioned previously in the introduction, all projects done within GNURadio Companion have an executable Python script as its output. All these scripts when executed produce a new window with graphical interface components, such as buttons, sliders, signals graphics among others, which are represented using the QT framework.

The main program is one script containing four buttons among other options, that open other four scripts. The base script is in charge of obtaining the base values of the spectrum power without drone activity. Spectrum and base scan scripts are in charge of

scanning in real time the spectrum power and compare with the base values. And the jamming script is in charge of generating jamming signals.

The scripts use GNURadio blocks and QT user interface components. Among the GNURadio components we have three custom ones. One is in charge of converting the data from the HackRF source into power with a Hamming window. Another is the power analyzer, which is in charge of creating the base powers file database with a specific format. The last one is the power comparator, which is in charge of comparing the real time power values with the base and create another file database with the comparison.

We will analyze in detail all the scripts and custom blocks in the next sections, giving us an insight on how things work behind the user interface.

3.1 Custom Block: logpowerfft_hamming

This block is in charge of converting the stream of data coming from the HackRF source into a power vector with a specific length, after applying the FFT with a Hamming window. GNURadio has a default block called Log Power FFT which does the same operations but with a Blackman-Harris window and it can't be changed. We prefer a Hamming window because it reduces the sidelobs compared to the main lobe, making it an ideal option for frequency selective analysis.

Basically we have copied the original block and applied a Hamming window. The input of this block is a stream of complex data and the output is a vector of numeric values which turns to be the power. The significant parameters to be configured are the sample rate, the vector length or fft size and the frame rate.

First it decimates the complex values stream coming at the sample rate parameter, into vectors with a rate specified in the frame rate parameter and a length specified in the vector length parameter. These values are transformed with the FFT and a Hamming window, then we obtain the squared magnitude of these complex values, and its respective logarithmic value with decimal base obtaining the power.

Since we have learnt previously that the sample rate is the bandwidth, what we do here is to transform the complex values stream into a 1024 (vector length/fft size parameter) points vector that occupy 20 MHz (sample rate parameter). So if we get to know which is the center frequency set in the source, we have 1024 power values for frequencies 10 MHz above and below it. That means a power value each 19,531 Khz for this scenario.

That concept is going to drive the logic behind the file database creation of the next two custom blocks.

The file involved in this block is *logpowerfft_win.py*.

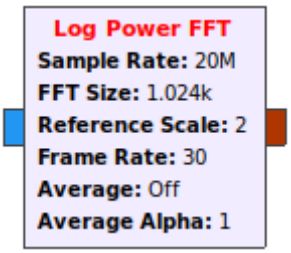

Figure 3.1: Default Log Power FFT block

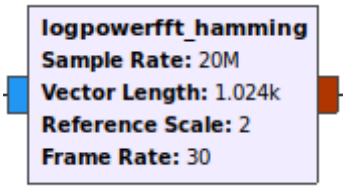

Figure 3.2: Custom Log Power FFT block with Hamming window

```
self._sd = blocks.stream_to_vector_decimator(item_size=gr.sizeof_gr_complex,
                                             sample_rate=sample_rate,
                                             vec_rate=frame_rate,
                                             vec_len=fft_size)

        fft_window = fft_lib.window_hamming(fft_size)
        fft = fft_lib.fft_vcc(fft_size, True, fft_window, True)
        window_power = sum([x*x for x in fft_window])

        c2magsq = blocks.complex_to_mag_squared(fft_size)
        self._avg = filter.single_pole_iir_filter_ff(1.0, fft_size)
        self._log = blocks.nlog10_ff(10, fft_size,
                                     -20*math.log10(fft_size)            # Adjust for number of bins
                                     -10*math.log10(float(window_power) / fft_size) # Adjust for
windowing loss
                                     -20*math.log10(float(ref_scale) / 2))      # Adjust for reference
scale
        self.connect(self, self._sd, fft, c2magsq, self._avg, self._log, self)
```

Code 3.1: logpowerfft_hamming operations code

3.2 Custom Block: power_analyzer

This is the block that performs the data operations to obtain the base power values and create a file database with them.

The input is a numeric vector of size specfied in the vector length parameter and the output is a document created in the specified directory and its name depends on the other parameters: sample rate, center frequency and vector length.

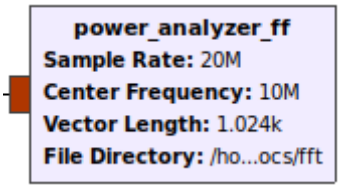

Figure 3.3: Custom power analyzer block

The name format of the output files is the following:

power_*(center_frequency)*MHz_*(sample_rate)*Msps_*(vector_length)*FFT.txt

So for the default values in the upper figure we have the creation of a document with the following name:

power_10MHz_20Msps_1024FFT.txt

A change in any of the three parameters will incurr in the creation of a different file.

The document has a value at the top of it indicating the number input vectors that have been analyzed. This value is know as the file index and is completely important since this document works with averages, as we will see later.

Below this value we will find as many lines as specified in the vector length parameter. Each line will have the following format:

(power)@(frequency)

Frequencies will have a six digit decimal precision and power will have two digit decimal precision. The power value is an average of all the powers received at a specific frequency. So a sample value will look like:

-80.61@2820.039101

Every time we have a new input vector, we open the current file and create a new one with the same file name format followed by a *_tmp* suffix. This newly created temporary file will have all the new calculated values. Once all calculations are finished, the current file will be deleted, since it will be outdated, and the temporary file will be renamed without the *_tmp* suffix replacing the deleted file. We follow this process to avoid a more difficult process of editing line by line an already created file.

To calculate the new values, we simply multiply the index value per the power, add the new power value and calculate the new average for every frequency.

```
73
-80.01@2820.000000
-80.95@2820.019550
-80.61@2820.039101
-79.81@2820.058651
-79.62@2820.078201
```

Code 3.2: Structure of a power file database

As it was stated before, a change in any of the three parameters generates a completely different document. This can be explained by the fact that we cannot make operations between a file that has 20 MHz distributed in 1024 points, with another one that has the same 20 MHz distributed in 2048 points. Frequencies wouldn't match to calculate the new averages. This principle is exploded in the base scan script since we need to create the power averages for all the spectrum possible, and in that case the varying parameter will be the center frequency.

It is important to make an emphasis on the novel structure of the file database proposed in this project. Normally the output of the sources, when written to a file, reach sizes of the order of the Megabytes in a few seconds, and in the order of Gigabytes for a few minutes, and its data contains complex numbers that are difficult to relate to. To handle files of these sizes in the analysis phase, we would need processing unit with high computation power. Since the objective of this project is to be executed in a Raspberry Pi with limited computation power, we had to create a way to store the data in a format in which the file size doesn't grow indiscriminately, and that at the same time it can be easily readable for a human. Both objectives are accomplished with this format, since the information of the file won't be increased at each iteration, it will only change the average values and the index, and at the same time can be easily interpreted when reading the data. Files for a vector length of 1024 will occupy around 20 Kilobytes, and for a length of 2048 its size will be around 40 Kilobytes, which make a it very efficient information storage method, because the file size will be almost the same for a scan of a few seconds, or scan of minutes or hours.

The file involved in this block is *power_comparator_ff.py*.

```
while not iterator.finished:
            current_freq = (iterator.index * self.freq_delta) + start_freq
            cached_power = 1000
            if file_exists:
                try:
                    cached_power = float(file.readline().split("@")[0]) #read power
                except Exception:
                    cached_power = 1000
            power = iterator[0]
            if cached_power != 1000:
                power = ((cached_power * file_index) + power) / (file_index+1)
            temp_file.write("%.2f@%.6f" % (power, current_freq/1e6))
            if (iterator.index != self.vlen-1):
                temp_file.write("\n")
            iterator.iternext()
        file.close()
        temp_file.close()
        os.remove(filename)
        os.rename(filename_temp, filename)
```

Code 3.3: Power operations and file replacement in the power_analyzer block

3.3 Custom Block: power_comparator

This block is in charge of receiving the real time data, compare against the base values and create a comparison file database for all frequencies.

The input is a numeric vector of size specfied in the vector length parameter and the output is a document created in the specified directory and its name depends on the other parameters: sample rate, center frequency and vector length.

For this block to work, it is mandatory that the base power values file database are created and in the same directory as the specified in the parameters.

This block has two woking modes that can be selected in the mode option: Percentage or Fixed Value. Both modes require a numeric value for them to work. We'll explain both modes later in this chapter.

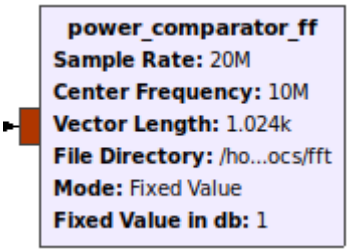

Figure 3.4: Custom power comparator block in mode fixed value

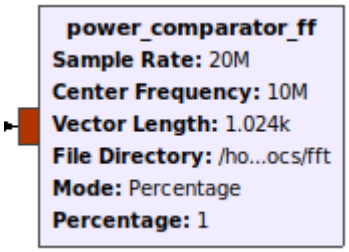

Figure 3.5: Custom power comparator block in mode percentage

The name format of the output file is similar to the created by the power analyzer and is the following:

compare_*(center_frequency)*MHz_*(sample_rate)*Msps_*(vector_length)*FFT.txt

So for the default values in the upper figure we have the creation of a document with the following name:

compare_10MHz_20Msps_1024FFT.txt

A change in any of the three parameters will incurr in the creation of a different file.

Identically like in the power analuzer, the document has a value at the top of it indicating the number input vectors that have been analyzed. This value is know as the file index and is completely important since this document works with averages.

Below this value we will find as many lines as specified in the vector length parameter. Each line will have the following format:

(value1);(value2);(value3);(value4);(value5)@(frequency)

Value 1 correpond to number of values above threshold.

Value 2 correspond to an average of values above threshold with respect to the file index.

Value 3 correspond to the minimum difference between all values and the threshold.

Value 4 correspond to the average difference between all values above threshold.

Value 5 correspond to the maximum difference between all values and the threshold.

The last value indicates the frequency to which at which all the previous values are calculated.

Values 1 to 5 will have a two digit decimal precision and frequencies will have a six digit decimal precision. So a sample value will look like:

21;0.66;0.31;5.16;10.20@60.273705

It means that 21 of all the values have exceeded the threshold, and it accounts for a 66% of all the received values. Of all the values compared, the minimum difference with respect to the threshold was 0.31db and the maximum 10.20db, and the average of all values above the threshold was 5.16db. All of this correspond to frequency 60.273705 MHz.

```
32
19;0.59;0.52;2.33;3.71@60.000000
17;0.53;0.18;2.22;4.36@60.019550
20;0.62;0.23;3.48;8.92@60.039101
22;0.69;0.22;3.17;9.14@60.058651
```

Code 3.4: Structure of a compare file database

When we have a new input vector, we open the base value file, the compare values file if exists, and create a new compare file with the same name format followed by a *_tmp* suffix. This newly created temporary file will have all the new calculated values. Once all calculations are finished, the current file will be deleted, since it will be outdated, and the temporary file will be renamed without the *_tmp* suffix replacing the deleted file. We follow this process just like in the power analyzer.

To calculate the values we have two different modes as explained earlier. Basically all the process will be similar except for the calculation of the threshold value. We read first the base power value and the previous comparation values if they exist, then obtain the threshold value and compare it to the input value. If the value exceeds the threshold, then we compare the values to the minimum (value 3) and maximum (value 5) and replace if necessary. Likewise, we calculate the average of the exceeded values (value 4) with the help of the file index,and increase the number of values above threshold (value 1) and its average (value 2). All of this is done for all frequency values in the input vector.

To calculate the threshold value in percentage mode, we simply calculate it multypling the base value per one plus the percentage value specified in parameters. So if the base value is 50 db and the percentage value is 10, we multiply 50 * (1 + 0.10) and obtain a threshold value of 55 db. The threshold value in fixed value mode is easier to obtain since we just add the value specified in the parameters with the base value, so if the base value is 50 db and the fixed value is 5, the threshold is 55 db.

```
            if self.mode == 1: #percentage
                threshold = cached_power*(1+self.diff_percentage/100)
            else: #fixed db
                threshold = cached_power+self.diff_db
            if power > threshold:
                exceeded_diff = power - cached_power
                exceeded_diff_min = numpy.minimum(exceeded_diff_min,exceeded_diff)
                exceeded_diff_average = ((exceeded_diff_average * exceeded_number) +
                        exceeded_diff) / (exceeded_number+1)
                exceeded_number = exceeded_number+1
                exceeded_diff_max = numpy.maximum(exceeded_diff_max,exceeded_diff)
            exceeded_average = exceeded_number/(file_result_index+1)
            temp_file.write("%.0f;%.2f;%.2f;%.2f;%.2f@%.6f" % (exceeded_number,
                        exceeded_average, exceeded_diff_min,
                        exceeded_diff_average, exceeded_diff_max, current_freq/1e6))
```

Code 3.5: Threshold and values calculations in the power_comparator block

Therefore each frequency hast its own threshold, which gives us flexibility, so we don't have an unique threshold value. The differences from the threshold are used so we can spot easily unusual peaks and also with the help of averages we can see how often values are above threshold for each frequency.

The novel file structure follows the advantages described in the power analyzer custom block. The file size for a 1024 vector length will be around 40 Kilobytes, and for a 2048 vector length around 80 Kilobytes. This proves to be a very efficient data storage method because the file size won't vary significantly if the scan lasts a few seconds, or for minutes or hours.

The file involved in this block is *power_comparator_ff.py*.

3.4 Script: Scan Base

This script is in charge of obtaining the average power values for all the spectrum range from 1 MHz up to 6 GHz.

The script is created initially in GNURadio Companion with three components which are the source, the custom log power block with Hamming window, and our custom power analyzer block. We also have six variables and four user interface components. The variables are defined to hold the values of the most important values such as the frequency, sample rate and directory. The user interface components will show the values of the previously mentioned variables, and we have and additional component that will control the time between frequency jumps. We'll explain how this work, later in this chapter.

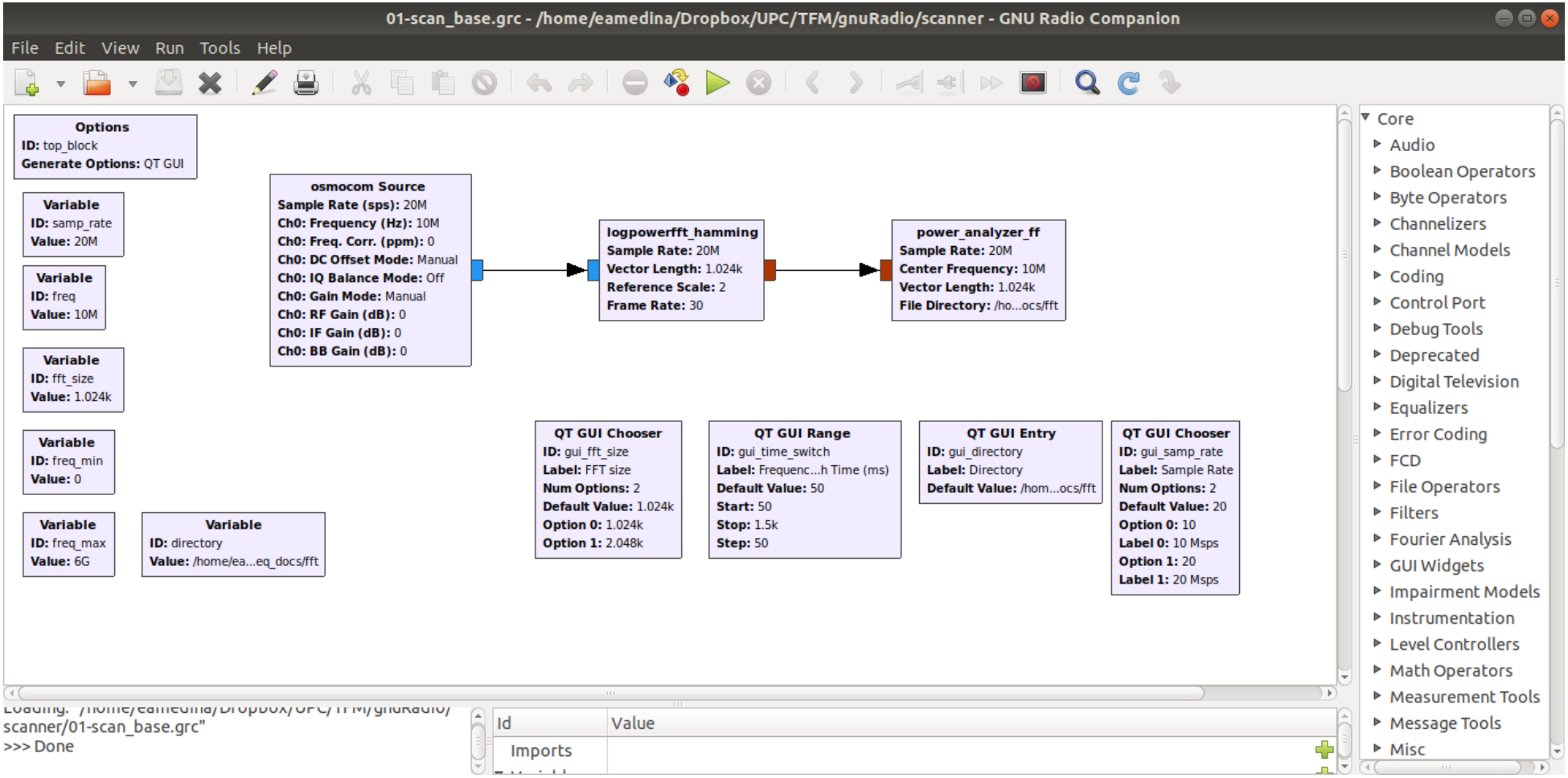

Figure 3.6: GNURadio Companion block structure for base scan

As it was explained in the GNU Radio Companion chapter, we have found several limitations that impede us to accomplish our objectives straight out of this software. We need to change the central frequency at which this script operates each certain amount of time, so we can cover all the supported spectrum by the HackRF, and the Companion software does not provide us with a block to do that. So, we must edit the script manually and we need a solid understanding of GNURadio blocks, QT and Python to do that.

Straight out of GNURadio Companion, the script indicates the central frequency to the source and it feeds with the data to the log power block, which decimates the data and converts it into a vector which size is determined by the vector length parameter. This output is fed into the custom analyzer block which generates the power values database file for that specific frequency, sample rate and vector length. It works correctly, but for one and only center frequency.

In order to adjust the script to our requirements of analyzing the spectrum from 1 MHz to 6 GHz, we need to create a process that changes the value of the center frequency every certain amount of time. For that purpose we choose a Timer from the QT framework, which changes the value of the frequency every certain time defined in the *time_switch* variable, and this process begins with the execution of the script.

The function in charge of changing the frequency is *recurring_timer*, which sets it in a value between the determined limits and dependent on the sample rate parameter. When we call the method *set_freq* we change the value of the frequency variable and inform both the source and the analyzer that the frequency has changed, so both can be synchronized and the analyzer can know to which frequencies correspond the values that it receives.

```
def startTimer(self):
    self.timer = QtCore.QTimer()
    self.timer.setInterval(self.time_switch)
    self.timer.timeout.connect(self.recurring_timer)
    self.timer.start()

def recurring_timer(self):
    if (self.get_freq()+self.samp_rate >= self.get_freq_max()):
        self.set_freq(self.get_freq_min()+self.samp_rate/2)
    else:
        self.set_freq(self.get_freq()+self.samp_rate)

def set_freq(self, freq):
    self.freq = freq
    self.osmosdr_source_0.set_center_freq(self.freq, 0)
    self.tfm_power_analyzer_ff_0_0.set_center_freq(self.freq)
```

Code 3.6.Timer in charge of switching the frequency

The only value that can be modified by the user is the frequency time switch, which is specified in milliseconds, and indicates the time at which the timer make the calls to change the frequency. We recommend a minimum value of 50 ms., since we have acknowledged that this time allows to have at least one input vector at the analyzer block per frequency jump. Using a sample rate of 20 Msps we must perform 300 frequency

jumps accounting for a total time of around 15 seconds for the whole spectrum range. This time can be greater, which means that we could have more than one reading at every centern requency, but the time to make a total spectrum loop will take longer.

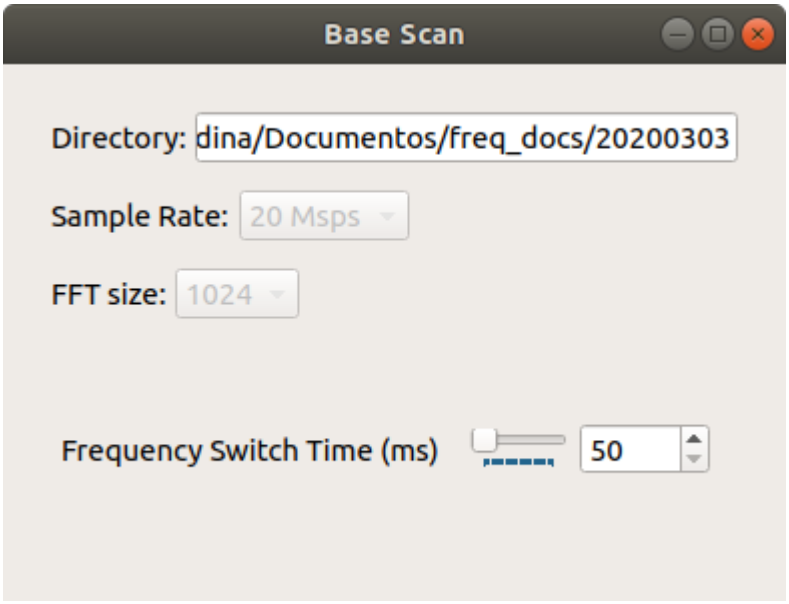

Figure 3.7: User interface components in base scan script

As we learned in the section of the power analyzer custom block**,** its output is a file with power averages whose name depends on the frequency, sample rate and vector length. Since in this script the frequency value is changing, the power analyzer block will generate as many files as frequency jumps. So, if it is the case of the previously explained example, we will have 300 files with the averaged powers for each determined frequency. All these files will account for a total size of around 6 Megabytes. This size won't vary significantly if the script runs for a few seconds or for hours, proving to be a very efficient data storage method.

All the files generated by this script will be fundamental to run the other spectrum analysis scripts and for the main script which displays graphics. Therefore it is mandatory to be the first executed script when using this project's software.

The files involved with this script are *01-scan_base.grc* and *01-scan_base.py*.

3.5 Script: Spectrum Scan

This script is in charge of comparing the real time power values with respect of the base power values for all the spectrum range from 1 MHz up to 6 GHz.

The script is created initially in GNURadio Companion with three components which are the source, the custom log power block with Hamming window, and our custom power comparator block. We also have seven variables and eight user interface components, to give the user a greater control over its execution. As in the previous script, the variables are defined to hold the values of the most important values such as the frequency,

sample rate, directory, and frequency switch time. Additionally we define variables to control the upper and lower frequencies limits, and also variables to control the operation mode of the comparator custom block. The user interface components will show the values of the static variables, and will provide components that will allow the user to control the time between frequency jumps, the upper and lower frequencies, and the comparator mode and value. We'll explain how this work, later in this chapter.

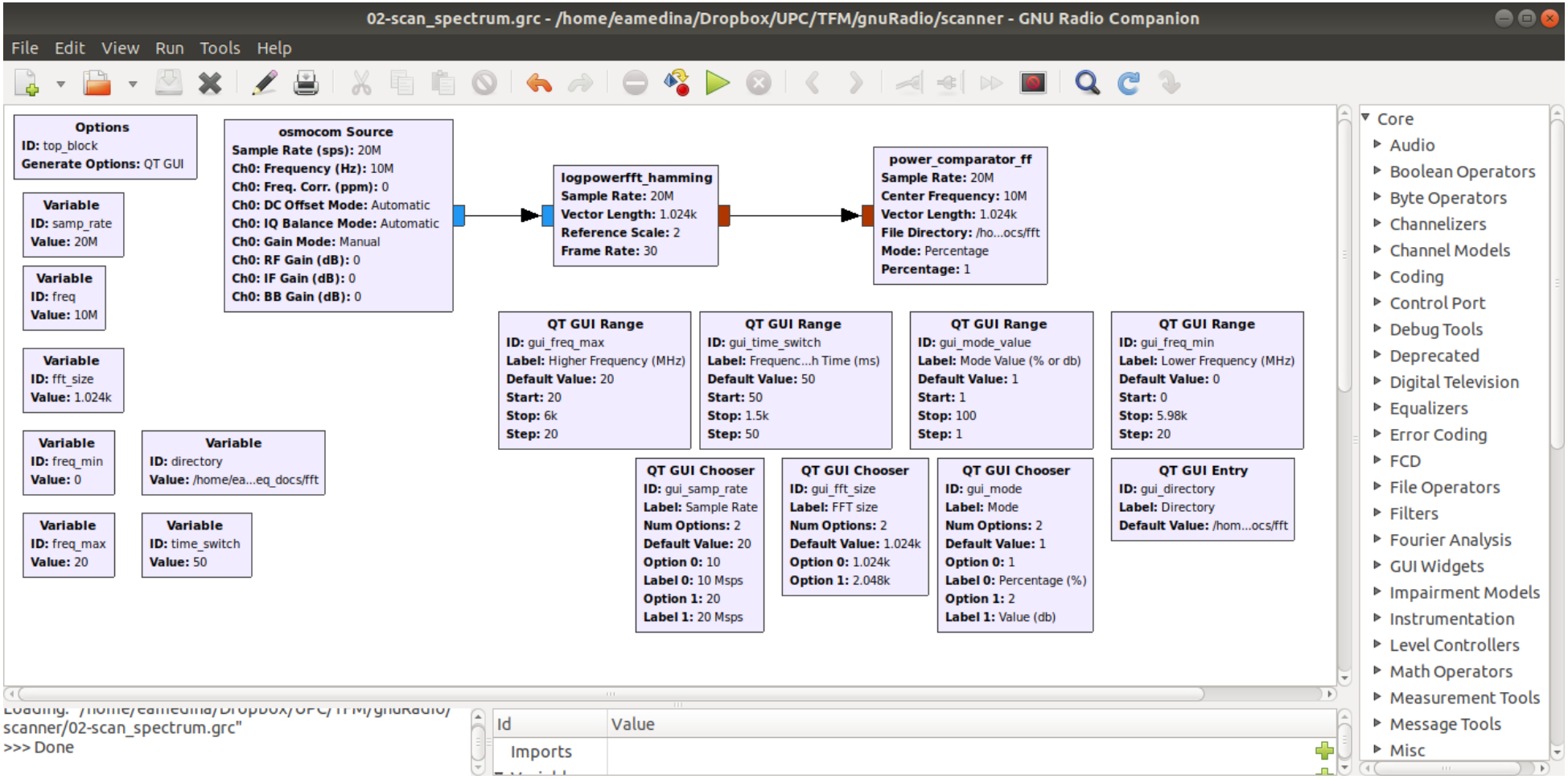

Figure 3.8: GNURadio Companion block structure for spectrum scan

Just like in the base scan script, we face the impossibility to create a process that changes the central frequency, so we have to do it manually modifying the script generated by GNURadio Companion.

The original script indicates the central frequency to the source and it feeds with the data to the log power block, which decimates the data and converts it into a vector which size is determined by the vector length parameter. This output is fed into the custom comparator block which generates the comparison values database file for that specific frequency, sample rate and vector length. It works correctly, but for one and only center frequency.

In this script we also analyze the spectrum from 1 MHz to 6 GHz, or user defined boundaries, so we need to create a process that changes the value of the center frequency every certain amount of time. We reuse the same solution that was used in the base scan script, choosing a QT Timer, which changes the value of the frequency every certain time defined in the *time_switch* variable, and this process begins with the execution of the script.

The functions that perform the task of changinf the frequency are the same as in the base scan script, and the code is the same as in the Code 3.6 block.

Besides using a different custom block as the file generator, in this script we give the user the chance to change more variables to her convenience. The user can change the value of the frequency switch time, the lower and upper frequency boundaries, and the operation mode and value of the comparator.

The frequency switch time indicates the value at which the timer executes the frequency changing function. For this script we have set a default value of 250 milliseconds, because the comparator has more operations to perform with the data and work with multiple files, and with this time we assure that we produce the corresponding file at every frequency jump. For a sample rate of 20 Msps, 300 frequency jumps will be performed accounting for a total time of 75 seconds to analyze the total spectrum from 1 MHz to 6 GHz. This time will vary if the user defines different boundary frequencies.

The user has the option to modify the lower and upper boundary frequencies, limiting the range at which the frequency jumps will be performed. This allows the user to focus on more specific frequencies rather than in the whole spectrum. These variables can change dynamically while the script is in execution providing flexibility to the program.

The comparator block has two operation modes that determine the threshold value at which power analysis will be performed. And this script gives the user the flexibility to change the threshold value while the script is running. The operation mode and the value can be changed at any time.

As we learned in the section of the power comparator custom block**,** its output is a file with differences and averages compared to the base values, and whose name depends on the frequency, sample rate and vector length. Since in this script the frequency value is changing, the power analyzer block will generate as many files as frequency jumps. So, if it is the case of the previously explained example, we will have 300 files with the differences and averages for each determined frequency. All these files will account for a total size of around 12 Megabytes. This size won't vary significantly if the script runs for a few seconds or for hours, proving to be a very efficient data storage method.

All the files generated by this script will be important to the main script which displays a graphic with both the base values and the maximum difference obtained from these files.

The files involved with this script are *02-scan_spectrum.grc* and *02-scan_spectrum.py*.

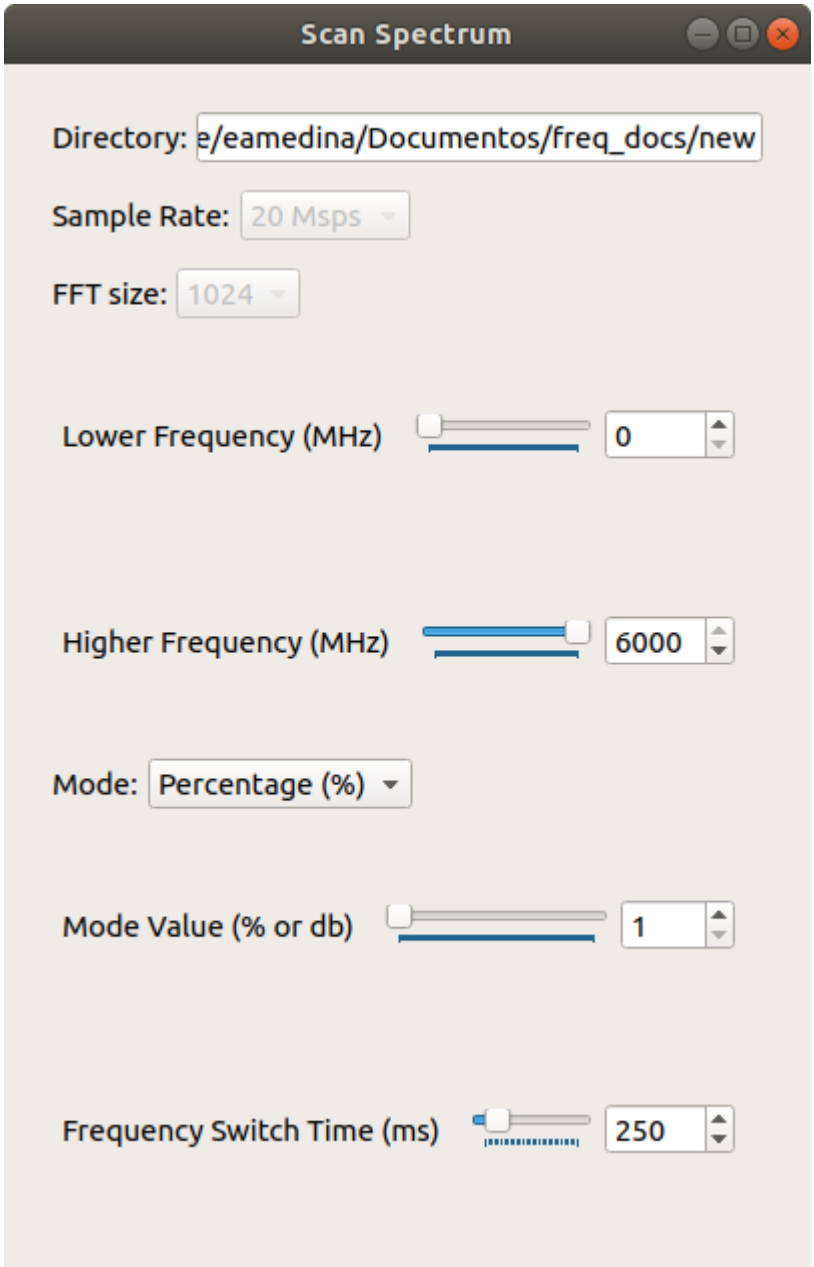

Figure 3.9: User interface components in spectrum scan script

3.6 Script: Band Scan

This script is in charge of comparing the real time power values with respect of the base power values for a limited bandwidth of up to 20 MHz and with graphical tools included.

The script is created entirely in GNURadio Companion with four components which are the source, the custom log power block with Hamming window, our custom power comparator block, and a native QT GUI Sink. We also have six variables and six user interface components. As in the previous scripts, the variables are defined to hold the values of the most important values such as the center and boundaries frequencies, sample rate, directory and operation mode and values of the comparator custom block.

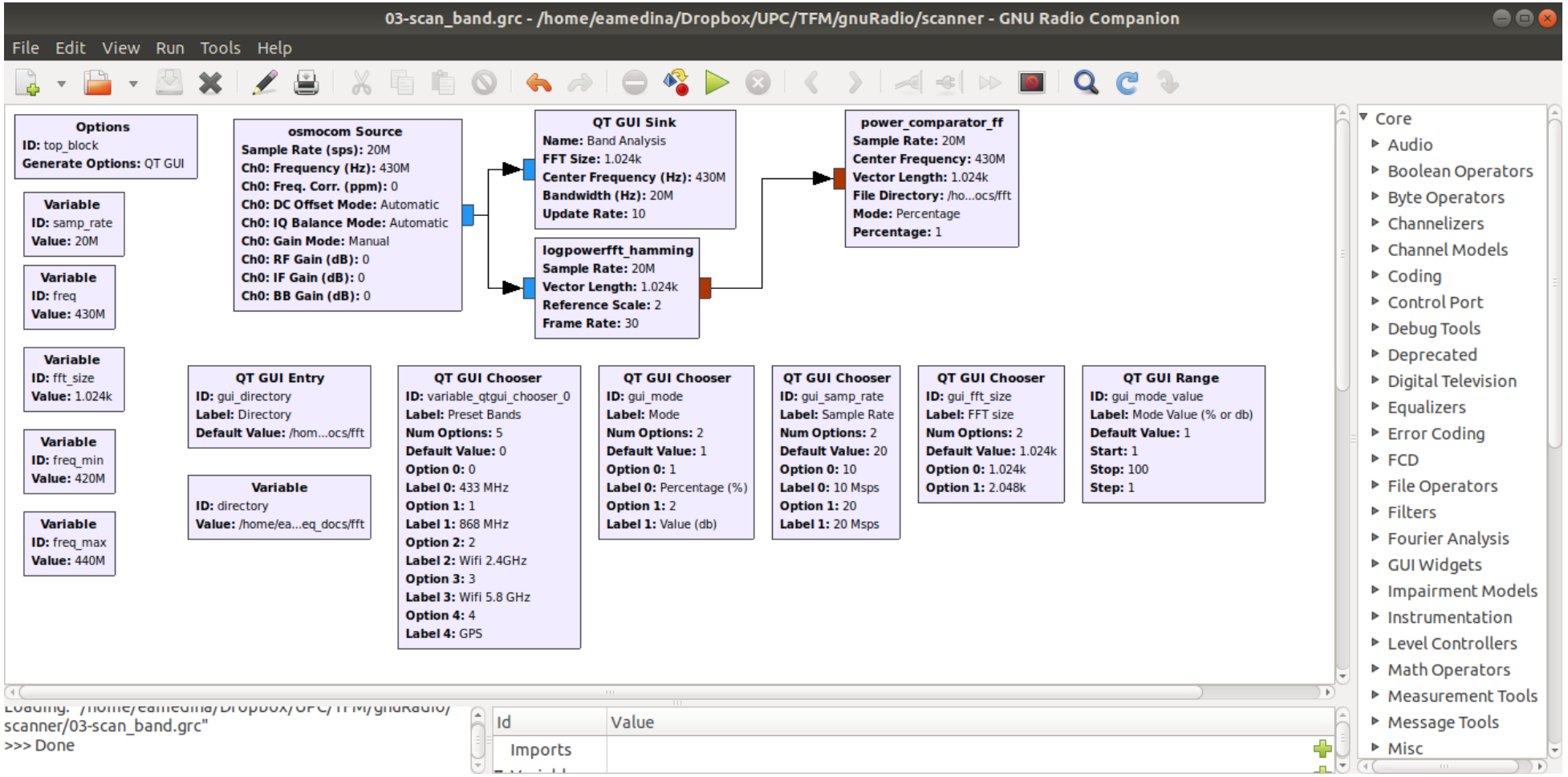

Figure 3.10: GNURadio Companion block structure for band scan

Contrary to the base scan and spectrum scan scripts, we won't have automatic frequency changes, we will give the user the total control of the frequencies analyzed when she want to focus its analysis in a given bandwidth. For that reason this script doesn't need to be modified.

The source feeds with the data to the gui sink and the log power block, which decimates the data and converts it into a vector which size is determined by the vector length parameter. This output is fed into the custom comparator block which generates the comparison values database file for that specific frequency, sample rate and vector length. The gui sink is fed directly from the source, and has options to display a real time frequency plot or a waterfall plot.

The operation mode selection and values for the comparator block follow the same guidelines as in the spectrum scan script.

Since this script is for a limited bandwidth which its maximum is given by the HackRF, we give the user an option to choose among the most common bands that can have drone RF activity, such as the ISM bands. This preset bands option sets the variables to predefined values of center frequencies dependent on the sample rate. We support only two sample rates, so for each option selected, we could have two different frequencies that work with its corresponding rate. One of the bands of interest is the 2.4 GHz WiFi band, and this band will be divided into chunks of a width determined by the sample rate, so for 20 Msps we can choose among five bands options, and for 10 Msps we can choose among ten bands options.

The main purpose of this script is to provide the user with a tool to inspect visually the band of interest. When using this script all the file database will also be generated, so all values obtained during the use of this script, will be reflected in the files.

The files involved with this script are *03-scan_band.grc* and *03-scan_band.py*.

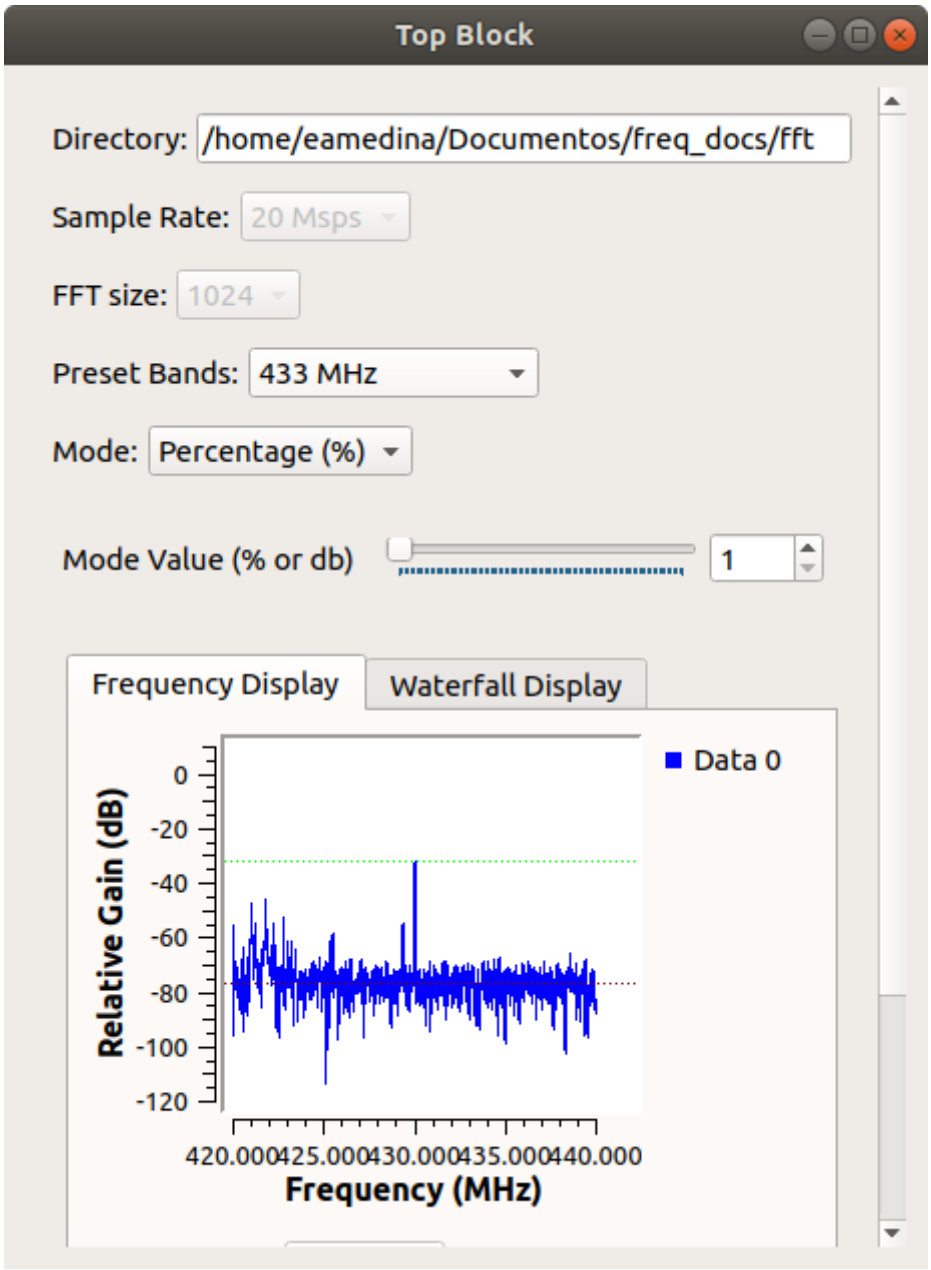

Figure 3.11: User interface components in band scan script

3.7 Script: Jammer

This script is in charge of generating a jamming signal of up to 20 MHz bandwidth in a frequency selected by the user.

The script is created initially in GNURadio Companion with four components which are the noise source, a multitplier, a throttle and the sink which in this case will be the HackRF. We also have four variables and nine user interface components. The variables are defined to hold the values of the center and boundaries frequencies, sample rate and frequency switch time. The user interface components willl give the user the opportunity to control the frequencies at which the noise will be emitted, and to control the HackRF gain parameters. We give the user the option to generate the noise in two modes, where

the central frequency can be static or dinamycally changed. We'll explain how this modes work later in this chapter.

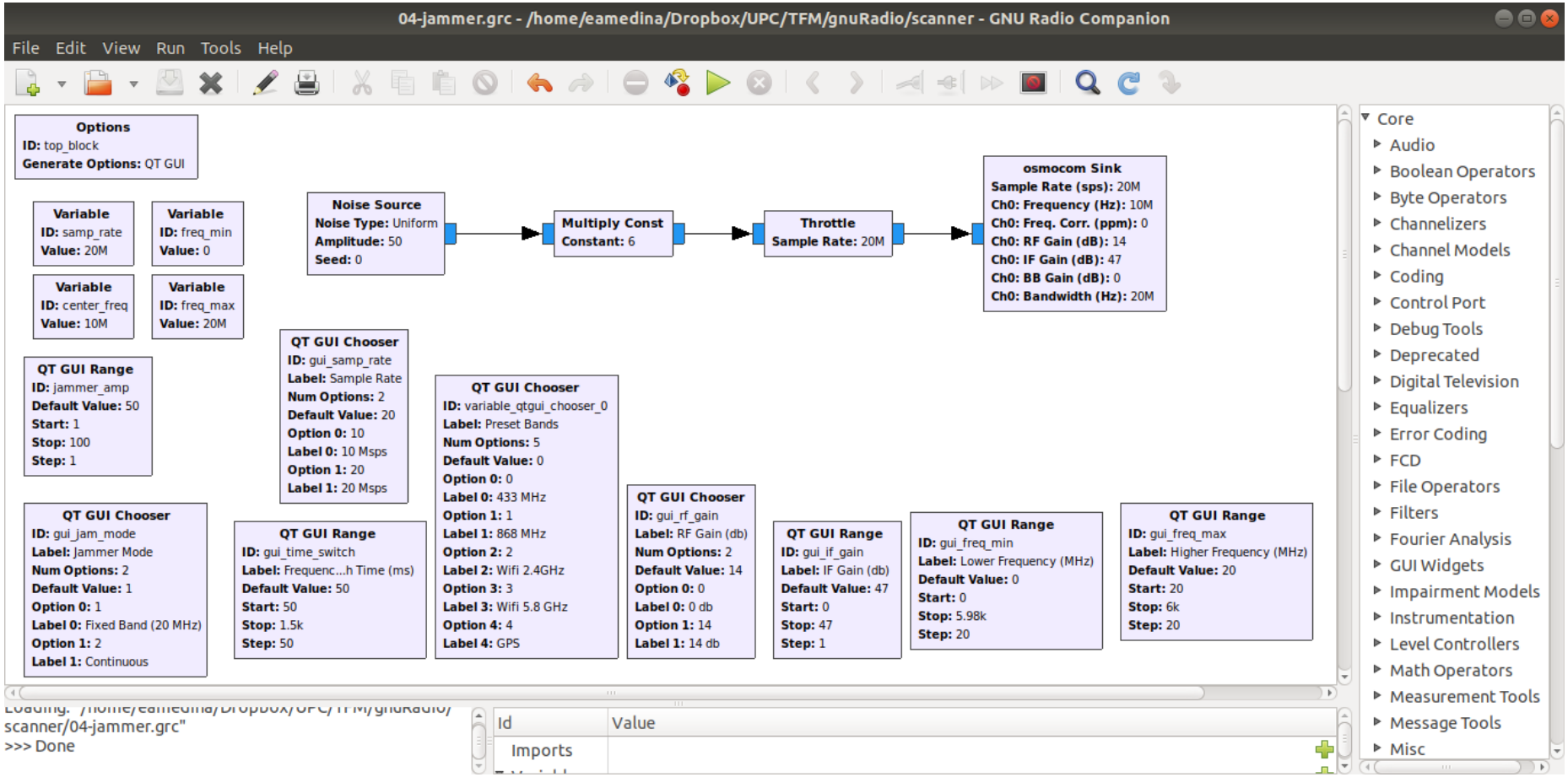

Figure 3.12: GNURadio Companion block structure for jammer script

Even though the script generated by GNU Radio Companion can work out of the box, we want to provide the user with the flexibility to generate the jamming noise to bandwidths greater than the maximum allowed by the HackRF, so we need to modify this script so we can achieve this functionality.

The original script consists of a noise source, that will use the CPU power to generate all the data, connected to a multiply block, that gives the source and additional gain, which is attached to a throttle block, which inoptimizes the use of the CPU power, and finally feed the sink which emitts all data. This solution works for a determined frequency and sample rate.

The limitation of 20 Msps as the maximum sample rate supported by the HackRF, gives us a 20 MHz maximum bandwidth that can we can interfere. But in bands such as 2.4 GHz, drones can use frequency hopping protocols along the whole band, so we must have the ability to interfere when this kind of beahviour is present. For that purpose, just like in the base and spectrum scan scripts, we need to create a process in which the frequency changes in time. We reuse the same solution that was used in the base scan script, choosing a QT Timer, which changes the value of the frequency every certain time defined in the *time_switch* variable, but contrary to previous scripts, this process will be initiated and terminated by the user exclusively.

We have two operations modes: a fixed mode with a bandwidth of the noise of 20 MHz, and a continuos one which boundary frequencies are limited by its corresponding variables and user interface components. When changing to the continuous mode, the timer will start to change the operating frequency of the sink, between the frequency boundaries and in a time that will also be controlled by the user with its respective interface.

Similar to what we saw in the band scan script, we provide the user with a set of preconfigured options based on the most common frequencies in which we can find drone activities. Additionally we provide a set of options corresponding to GPS frequencies, so we could interfere the communication of drones operating in autonomous mode.

The files involved with this script are *04-jammer.grc* and *04-jammer.py*.

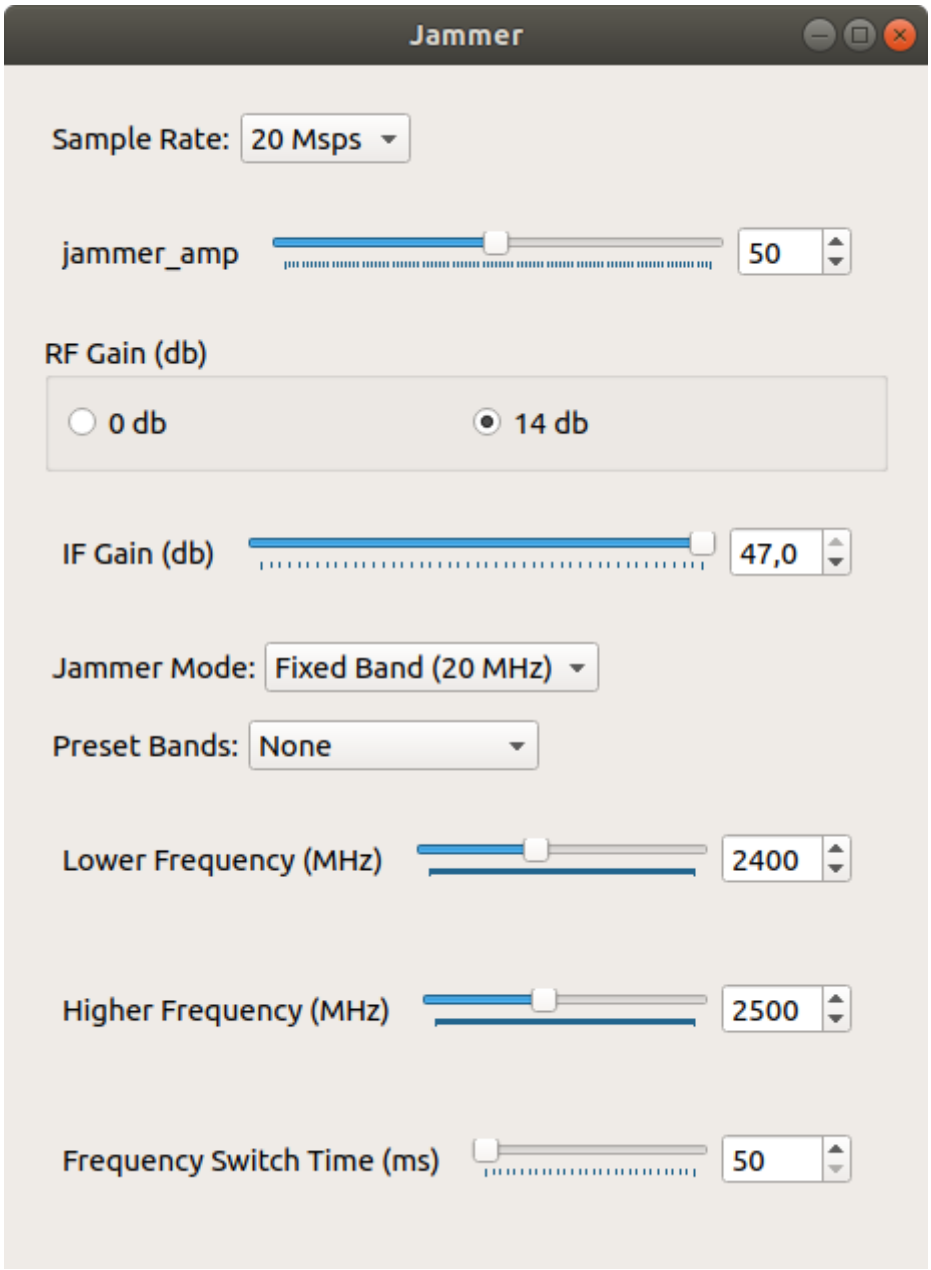

Figure 3.13: User interface components in jammer script

3.8 Script: Main Program

The main script of the project is the controller window. It has four buttons that execute the data acquisition and analysis scripts, and the inhibition script. It has three fields to configure the main parameters that will be passed to other scripts and will be used to display graphics within. These parameters are the directory, sample rate and FFT size.

In this project we are working with a file database approach, so that all information gathered and produced by our scripts will be stored in files that are located in the directory specified in the parameters. At the same time, these files depend on the values of sample rate and FFT size. As we will learn in the following sections, the change in one or both of these parameters lead to creation of different files, so we could have different combinations of generated files.

In this main window, we also have a graphic section, a table and an additional option with the “Choose band” label. When there is no data in the directory, it shows an empty graphic and an empty table. If there is data, we see a graphic displaying a line with the amount of points specified in FFT size parameter, in an interval of frequencies specified in the sample rate parameter. The blue line shows the data obtained from the Base Scan script, which are time-long averages of the power received in that frequency band. The red line shows the data obtained in the Spectrum Scan and/or Band Scan scripts, specifically the maximum power values obtained in that frequency band.

The table below shows all the data processed from the Spectrum Scan/Band Scan scripts. For a better understanding of the data here displayed, please go to those scripts sections. The table shows all the frequencies measured, from 1MHz to 6GHz, with the processed data. Since we are dealing with differences from the base power values, here we display those values, ordered by default by the maximum difference value. All columns of this table can be reordered ascendent or descendent with a click in the respective header.

The “Choose Band” parameter is set by default in the “CONTINUOUS” option. This means that in the background a timer will change the data displayed in graphic every 3 seconds. If the sample rate is set to 20 Msps, it will display sets of 20 MHz, changing the limits every 3 seconds. It also has an option for “ALL”, which will display all the data from 1MHz to 6GHz. As it is does not provide a sharp look at the data, extra controls are given so that the user can change the frequency to be displayed and the bandwidth of the graphic. It is important to understand the file generation process of the base and spectrum scan scripts to select with accuracy the parameters that will generate a graphic in this option. Other options available in the “Choose Band” are the most used bands for drone communications which are 433 MHz, 868 MHz, and the Wifi band. This provides an easy configuration to check the beahviour of the spectrum graphically in bands of interest.

The purpose of the development of these two tools is to help the user to identify uncommon signals both visually and statistically, so the user can focus on analysing a specific band and confirm the presence of a drone.

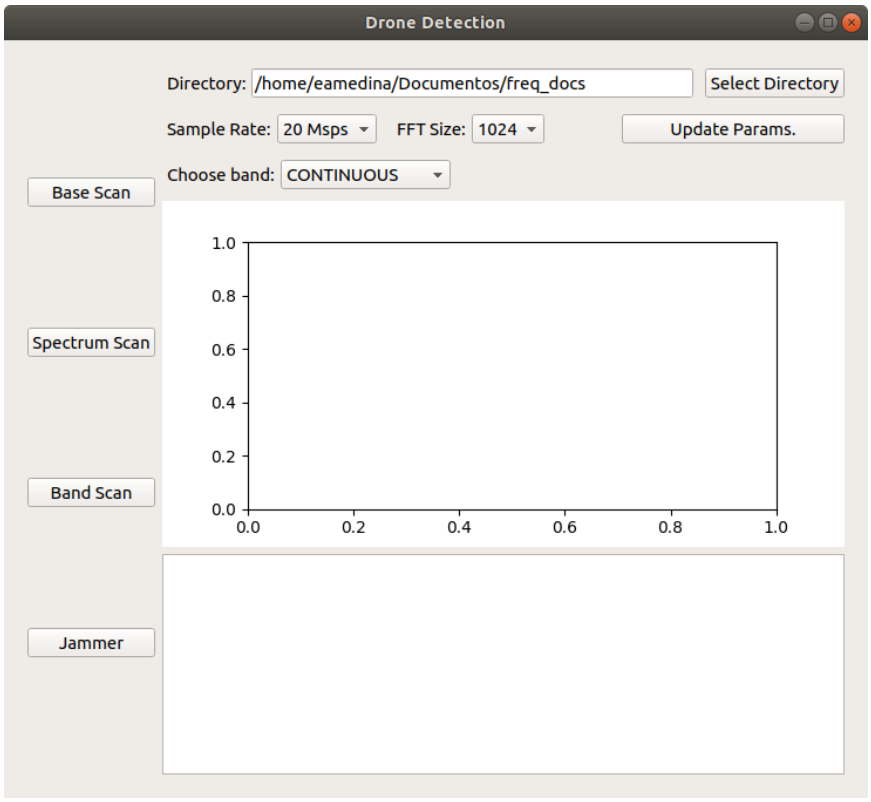

Figure 3.14: Main script with no data

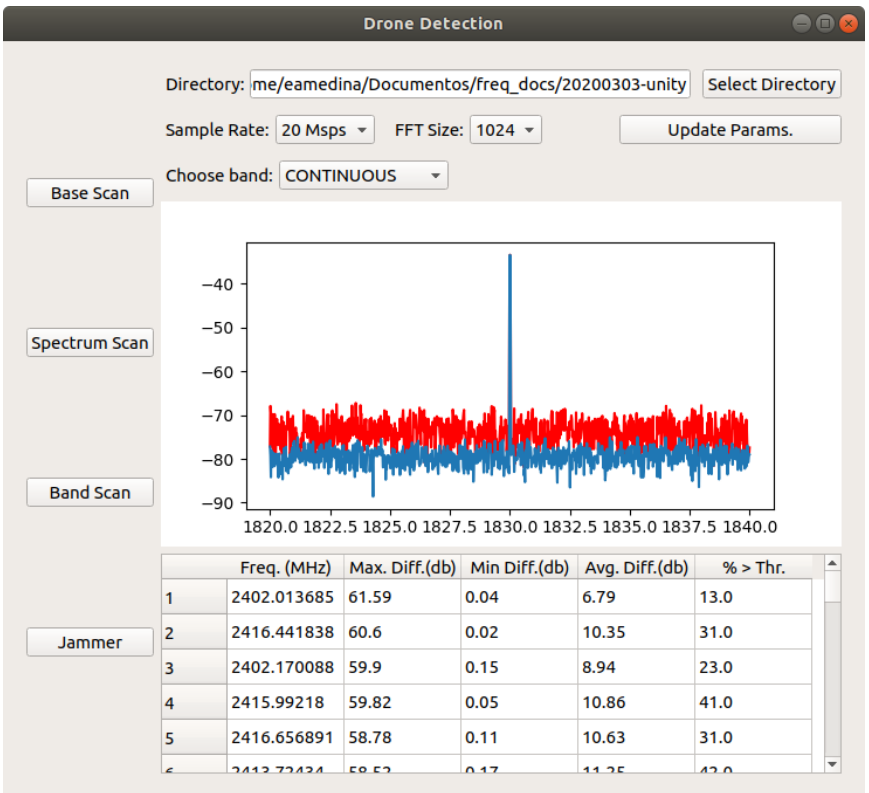

Figure 3.15: Main script with data in continuous mode

4. Results

TODO

5. Budget

Depending on the scope of the thesis, this document should include:

- Components list with approximate costs (prototype).
- Design, prototyping and other tasks costs (hours_person x cost).
- Financial viability analysis.

6. Environment Impact (optional)

The tasks involved in the completion of this thesis and the results having any identifiable environmental impact should be described in this section.

The impact may be negative (environmental cost), positive (solution that improves on the environmental impact of other projects) or both.

7. Conclusions and future development

This should include your summary, conclusions and recommendations.

Bibliography

A thorough reference list such as that shown in the following examples: Conference paper [1], journal paper [2], book [3], standard-1 [4], standard-2 [5], online reference [6], patent [7], M.S. thesis [8] and Ph.D. dissertation [9].

[1] J. Polastre, R. Szewczyk, D. Culler. "Telos: enabling ultra-low power wireless research". *In Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005*, 25-27 April 2005, Los Angeles, USA. pp. 364-369. doi: 10.1109/IPSN.2005.1440950.

[2] V.C. Gungor, B. Lu, G.P. Hancke. "Opportunities and challenges of Wireless Sensor Networks in Smart Grid". *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 3557-3564, October 2010. DOI: 10.1109/TIE.2009.2039455.

[3] R. Faludi. *Building Wireless Sensor Networks: with ZigBee, XBee, Arduino, and Processing*, 1st ed. Sebastopol, USA: O'Reilly Media, 2010.

[4] *Internet Protocol, Version 6 (IPv6) Specification*. IETF RFC 2460, December 1998.

[5] *IEEE Standard for Information technology. Telecommunications and information exchange between systems Local and metropolitan area networks. Specific requirements. Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Std 802.11-2012.

[6] T. Tarun, P. Viswanathan, S. Suman. "Wireless Sensor Network White Paper". *Tetcos Engineering*, 2012. [Online] Available: http://www.tetcos.com/Enhancing_Throughput_of_WSNs.pdf. [Accessed: 23 October 2012].

[7] J. P. Wilkinson, “Nonlinear resonant circuit devices,” U.S. Patent 3 624 125, July 16, 1990.

[8] M.V. Alvarez Fernández. "Feasibility study and design for Wireless Sensor Networks in a space environment". M.S. thesis, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Delft, The Netherlands, 2011.

[9] J. O. Williams, “Narrow-band analyzer,” Ph.D. dissertation, Department of Electrical Engineering, Harvard University, Cambridge, MA, USA,1993.

Appendices (optional)

Appendices may be included in your thesis, but it is not a requirement.

Glossary

A list of all acronyms and what they stand for.

LINKS

dji firefighting - https://venturebeat.com/2019/04/20/dji-rd-head-dreams-of-drones-fighting-fires-by-the-thousands-in-aerial-aqueduct/

heat drone - https://abcnews.go.com/US/drone-finds-missing-year-minnesota-boy-cornfield/story?id=66327536