UNIVERSITÀ DEGLI STUDI
DI PERUGIA

DEPARTMENT OF ENGINEERING

Master's thesis in.

COMPUTER AND ELECTRONIC ENGINEERING

Microwave Noise Measurement Using Software Defined Radio: Experimental Characterization, Calibration, and Radio Astronomy Applications.

Speaker
Prof. Federico Alimenti

Candidate
Francesco Alunni

Academic Year 2023/2024

Dedicated to my mother,
Simonetta

Index

List of figures

List of tables

Chapter 1

Introduction

1.1 Introduction

In this thesis we want to talk about the study, preparation, and demonstration of being able to do radio astronomical measurements of celestial bodies (in our case, the sun) in the 10.75GHz-11.75GHz band (thus within the X-band) using econo- mical and easily available components, such as a SDR (Software Defined Radio) and a satellite dish. This study is`e done in an attempt to find a nè economi- c and compact alternative to the analog backend developed in previous years, to allow its use in applications where space` is limited, such as within a CubeSat. The im- portance of the use of a microwave radiometer in radio astronomy applications`is considerable , since in the microwave band (1-300 GHz) important phenomena can be observed for the study of celestial bodies, such as the study of the sun's flare, as well as the study of the atmosphere and seas, as will be dhnè later, because of the emission of water and hydrogen within this band (Specifically, the L 1-2 GHz band). In fact, by measuring the power emitted in the microwave band, one can detect the powers emitted by the various elements, thus being able to derive their absorptions over each frequency band, which help to study their behaviors, even at the quantum level[15]. Specifically, we`and tried to use a system composed of an 80cm diameter satellite dish, a generic LNB to be able to receive the signal, amplify it and translate it down in frequency so that it can be received by an RTL- SDR, which`and an inexpensive Software Defined Radio born from the hardware modifications of the DVB-T receiver, which connected to a PC allow it to act as a radio scanner in any frequency between 24MHz and 1.7Ghz or so, then defining via software through- towards GNU Radio all the digital filters that exploit the signal to calculate its noise power in the 2MHz band centered at the 1GHz frequency. My task in this project`was mainly to think about the part of the system from the LNB to the PC, making sure that all components were well characterized in order to perform mi- surations accurately, looking at the pò of using the LNB to make it à flexible as possible in frequency, characterizing the SDR and implementing the script

on GNU Radio that receives as input the I/Q signals (in phase and phase quadrature) digitized by the SDR, from which it calculates its in-band power and its equivalent noise temperature, as well as writes them in a formatted manner to a text file, interpretable by the GNUPlot software to plot the various graphs. It also`and had to implement a basic calibration script (based only receiving blackbody emission) to avoid às much as possible drifts in temperature, a problematic feature often found in such inexpensive devices.

1.2 State of the Art.

The following discusses the current state of the art, cio`and previous work prior to this of both scientific and educational relevance.

1.2.1 ESA-CIMR

The current state of the art`and the Copernicus Imaging Microwave Radiometer (CIMR), two satellites that will be launched sequentially with each equipped with a multifrequency microwave cone scan radiometer to measure sea ice concentration, sea surface temperature, ìy and more. The surface of the globe hwillbè capable of measuring ìd by 95 percent. The first satellite will`a launched in 2029.

Specifically, the frequency bands that can be measured with the radiometer are[6]:

- L-band (1.4135 GHz): allows measurements of ìy sea, soil ìne of the surface of the and winds over the ocean.
- Band C (6.9GHz): allows measurements of sea surface temperature, land and ice, as well as supporting sea-ice parameters.
- X-band (10.65 GHz): also allows measurements of sea surface temperature, land and ice, as well as supporting sea-ice parameters.
- K-band (18.7 GHz): allows measurements of sea, ice and snow parameters.
- Ka Band (36.5 GHz): also allows measurements of sea, ice and snow parameters.

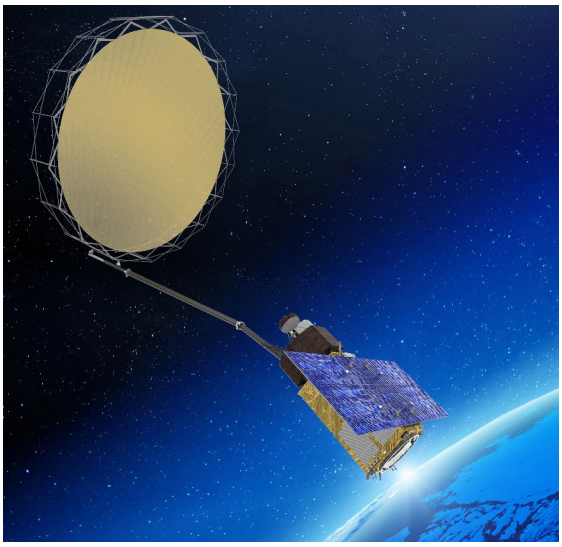

Figure 1.1: Copernicus Imaging Microwave Radiometer[6]

1.2.2 Radiometers operating in L-band using an SDR

Searching through scientific articles, one can find relatively recent studies about the use of SDR as the main backend of radiometers operating in the L-band (1-2 GHz), a very important band for studying much of the phenomena related to water, [illegible] and hydrogen general, having within, on the 1.4 GHz, the frequency where the i-Doprogen emits m[illegible] at the electrmoagnetic level. Among the articles y[illegible] find one describing FMPL-2 (Flexible Microwave Payload-2), a passive SDR-based microon- de instrument mounted on ESA's FSSCat mission. This instrument occupies one of the [illegible] of the CubeSat used for this mission and covers two roles: that of reflectometer-GNSS (Global Navigation Satellite System) and that of radiometer in ban-da L, whose main use`is to make measurements of geophysical parameters, such as the coverage of ice versus water on the earth or of soil m[illegible]The instrument`is mainly composed of an FPGA (Field Programmable Gate Array) that virtualizes a dual-core ARM processor that takes care of the calculations and an SDR that takes care of I/Q [10]. data demodulationAnother interesting paper` is the one describing the architecture of a radiometer operating in L-band exploiting both linear, vertical and horizontal polarizations, which`and mounted on a UAS (Unmanned Aircraft System, i.e.`and a drone). The purpose of this`radiometer is to effet- tuate measurements of soil and water bodies in agricultural settings, exploiting drones, thanks to the [illegible] of measuring both polarizations and taking advantage of a 30MHz SDR sampling rate to be able to demodulate the I/Q components, immediately converting them into brightness temperature (brightness temperature) T_B obtained through internal and external calibrations[7].

1.2.3 Calibrations through use of satellite beacons.

One of the mò important steps when it comes to making the mò accurate measurements possible,`and calibration, which gives a reference point for the instrument to accurately measure the quantity of interest. In our case, to calibrate an instrument that is to measure the equivalent noise temperature of a celestial body from the- the power it radiates in the observed frequency, we measure the electromagnetic emission of a black body, whose temperature`is known and defines the so-called $T_{hot}$, that`and the reference hot temperature which, in conjunction with a target of low temperature T_{cold}, one can solve the equations that determine real gain (gain) of the system and its noise figure, in effect characterizing the system to reduce the systematic-type error. In the article cited above, in addition to using a RAM absorber panel as the hot body and the sky (which is usually between 60-70K) as the cold body to characterize the instrument, it discusses using CW (Continuous Wave) beacons from geostationary satellites, which operate on various microwave frequencies (from 11 GHz up to even 40GHz), to determine the attenuation and noise of the sky, taking away an additional disturb- bo to radio astronomy measurements due to the presence of the atmosphere[1]. Calibration through T_{hot} and T_{cold}` is one of the best, due to the fact that one has to solve a system of equations with two unknowns $G_{(LNB)}$) and $T_{LNB}$, and thus by avoiding making assumptions, one can easily characterize the mò important stage in the chain (i.e.`and the first one). A demonstration based on the thermal noise formulas à be found in A.2.

Aò see with the study of the state of art, there is still no real X-band and K_U radiometer composed of cost-effective components such as the one that this thesis sets out to do, whose total cost of such a system, consisting of SDR, LNB, Bias-T and satellite dish,à amount to around 80€, making it particularly inexpensive, effective and flexible, given the fully digital nature of the SDR, which makes it dynamically configurable and makes the signals detected by it completely manageable by computer.

Chapter 2

Materials used

The following will discuss the various materials, both software and hardware, used in order to obtain the desired measuring instrument.

2.1 SDR

Software Defined Radio (SDR)`and a digital radio frequency device capable of acting as a radio receiver (or spectrum analyzer) by managing data via software, tra- mitting digital filters to the digital signal obtained. Specifically, the RTL-SDR
and a hardware modification of the DVB-T television receiver, in which a bug was discovered that allowed I/Q components to be received via USB interface in addition to the p of setting the tuning frequency far beyond the UHF television bands.`and composed mainly of two blocks:

- The Rafael R820T Tuner, i.e.`and a superheterodyne receiver composed as seen in Figure 2.1
- The Realtek RTL2832U COFDM (Coded Orthogonal Frequency-Division Multiplexing) demodulator, a device capable of demodulating the signal received from the tuner and sending its I/Q components via USB, making them receivable by a PC.

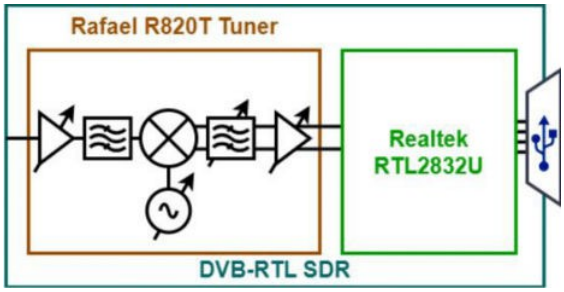

Figure 2.1: Block diagram of the SDR[13]

The tuner, as mentioned, follows the principle of the superheterodyne receiver, which consists of using a local oscillator (LO) that generates a sine wave of varying frequency, which multiplied with the input signal through a mixer, by the theory of modulated signals, frequency shifts the signal up to a certain frequency, called IF (Inter- Frequency). When multiplying a signal of a certain *B-band* and center frequency f_{IN} with a frequency sine wave f_{LO}, we get:

$$f_{(IF)} = f_{IN} - f_{LO} \tag{2.1}$$

Whose frequency f_{IF} is the frequency of interest, being the center frequency of the IF bandpass filter, placed subsequent to the mixer in the chain, whose by determines the frequency resolution of the spectrum analyzer (i.e. and the SDR). This frequency-shifting operation is done continuously by linearly varying the frequency of the LO with a ramp generator, thus making the entire signal "pass" through the IF filter and therefore or show each component of the signal. Next, the signal is sent to the Realtek RTL2832U chip, which converts the signal to digital and then demodulates it for sending the I/Q components via USB to the PC, which then handles the data with GNU Radio to make measurements. In addition to these building blocks, the mixer and anticipated by a low-pass filter, which reduces the signal to the receiver band, and signal amplifiers and attenuators to condition the signal in such a way as to make its dynamics manageable by subsequent devices, especially the ADC (Analog to Digital Converter) of the demodulator chip, which and optimal if the signal varies throughout the input dynamics. Other RTL-SDR receiver features of interest, obtainable on the RTL-SDR website[12] are:

- Frequency range from 24MHz to 1766 MHz, with attenuation determined experimentally after 1200MHz
- Sampling rate of 2.56 MS/s stable, which à go up to 3.2MS/s with risk of sample loss, risky in case you want to demodulate or decode signals. Interesting to note that if you use a sampling rate f_s, sampling a complex signal (i.e. and the I/Q signal), you are able to sample a signal with $f_{max} = f_s$, per complex sampling theory.
- It has an input impedance of 75 Ω, being in principle a UHF te- levisive receiver, so by connecting through an SMA coaxial cable with a characteristic impedance of 50 Ω, a small mismatch is achieved, but resulting in a loss of only 0.177dB, easily approximated.
- The resolution the ADC and 8bit, although experimentally 7 effective bits of resolution have been determined, which always depends on the type of signal and its probability a in taking certain values over others, since ADCs

are designed with in mind a signal with a`uniform probability of taking on any value within the dynamic range, which is not`and always true.

Figure 2.2: SDR[13]

2.2 LNB

The LNB (Low Noise block)`is a receiver device that is mounted at the focal point of the television satellite dish and serves mainly to amplify the received signal and lower it in frequency through a mixer to a frequency manageable by a simple coaxial cable without suffering too much loss, since satellites transmit in the microwave band and to transmit them after receiving them would require a waveguide system. Mainly the LNB`and composed of:

- LNA (Low Noise Amplifier), amplifier designed to be low noise figure, being the first stage of the amplifier chain which, by Friis' formula proves to be the most important in determining the noise` a of the whole chain. Thus reducing possible contributions from later noisy devices within the system.
- Mixer and local oscillator (LO) to perform fixed signal downconverting to a` lower band, to make the signal more manageable through use of more simple and inexpensive devices such as, in our case, the SDR which has a Maximum bandwidth of only 1.7 GHz.
- Hornfeed antenna, a type of horn antenna often used to channel reflected microwave electromagnetic waves from the dish to the focus point, where the said antenna is located.

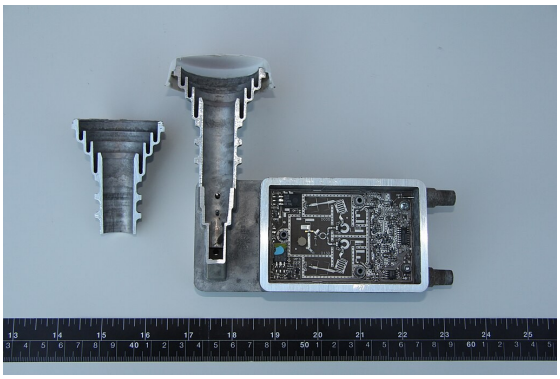

Figure 2.3: Cross-section of an LNB[3]

In our case, we used a European universal LNB (also called "Astra" LNB, because of their wide use with Astra geostationary satellites) whose main characteristics are:

- Local 9.75 GHz and 10.60 GHz oscillator for downconverting the 10.70-11.70 GHz and 11.70-12.75 GHz bands in the IF (Intermediate frequency) range 950- 1950 MHz and 1100-2150 MHz respectively
- Linear polarization selectable between vertical and horizontal through the feed tension, 12-13V for vertical polarization and 18V for horizontal polarization. Linear polarization is used in this band instead of circular polarization because of the simplicity`a of antenna implementation.

These features make this LNB usable on 4 different bands, depending on the combination of LO and polarization used. Thanks to this device`and it is possible to measure the X-band and K_u band without receiving much input noise and being able to use inexpensive RF receivers. [3]

2.3 Bias-T

The Bias-T`is a circuit used to separate the RF signal components from the DC component,`and used to power the LNB without bringing this receiving component into the SDR, since it result in noise and spurious frequency generation.

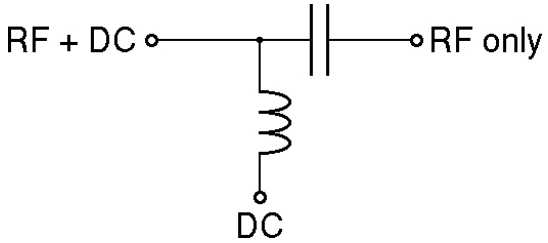

Figure 2.4: Bias-T[9]

Specifically, to understand the frequency behavior of the Bias-T, we need to observe the frequency response of its components, i.e.`and the capacitor, which acts as the DC block, and the inductor, which acts as the RF block. In the case of the capacitor, its impedance`is defined by:

$$Z= \frac{1}{j2\tilde{A}f\,C} \tag{2.2}$$

Where wà observe that by making the limit for f tending to infinity we get:

$$lim_{f\to\infty} \frac{1}{j2\tilde{A}f\,C} = 0 \tag{2.3}$$

This`and, as the frequency increases, the impedance decreases, gæloser and closer` to a short circuit for radio-frequency signals, while for the DC component (f= 0) there is:

$$lim_{f\to 0} \frac{1}{j)2\tilde{A}f\,C} =\infty \tag{2.4}$$

Therefore`or becomes an open circuit for the DC component. In the case of the inductor, however, we have:

$$Z= j2\tilde{A}f\,L \tag{2.5}$$

Where, making the same limits as for the capacitor, we get:

$$lim_{f\to 0} j2\tilde{A}f\,L= 0 \tag{2.6}$$

$$lim_{f\to\infty} j2\tilde{A}f\,L =\infty \tag{2.7}$$

Therefore`o the inductor has dual frequency behavior, acting as a short circuit for the DC component and as an open circuit for the RF component.

The bias-T`and particularly important in this system, as it is necessary in order to feed the LNB with the desired DC voltage, without it being received by the SDR , which only wants RF components, and avoiding the reverse, to reduce signal losses.

2.4 GNU Radio

GNU Radio`is a free, open-source development software that allows Software Defined Radios to be imple- mented through the use of digital signal processing blocks. It à be used using low-cost external RF hardware sources or at- traction through internal software simulations. It has wide use in many fields, including research, industry, and hobbyist fields in order to dis- support both wireless communications research and real-life radio systems. This software easily allows, through block design, the implementation of filter systems and other signal ge- stion blocks in a purely digital manner, making it very easy to create

software-defined radios. Although it has a large number of blocks to cover most of the field's implementation needs, it also allows scripts to be developed in Python to be able to implement any`that does not pre-exist, making it extremely flexible in signal processing. The software`also comes with an online wiki that particularly accurately describes all the blocks, spe- cifying all the inputs, outputs, and parameters that can be set and for what purpose, in order to best utlize each block depending on the use of interest.[14]

Figure 2.5: Logo of GNU Radio[11]

2.5 GNU Plot

GNUPlot` is an open-source graphical software with the main purpose of allowing students and scientists to graphically see mathematical functions or experimentally collected data in order to analyze the data, showing them in the appropriate.[8] wayIt is used by programming script files that are then executed in order to show the processed data, using its own syntax. An example of the code used in this thesis to show graphs of data obtained from measurements is shown below:

```
#!/usr/bin/gnuplot -persist

set terminal qt 0 #Create a new window
set title "Power measurements "#I set the title of the graph
#I set the axis titles
set xlabel "Time (s)"
set ylabel "Temperature (K)"
set grid #Most grid on graph

#Graph 5 data sets on the graph, overlaid with different colors.
    →to show the differences
plot [0:1000] "0.1s data with gray moving average.txt" - with lines title
    '→   "0.1s with moving average"

replot [0:1000] "0.3s data with gray moving average.txt" using "time:"
    →temperature" with lines title "0.3s with moving average"

replot [0:1000] "1s data with gray moving average.txt" using "time:"
    →temperature" with lines title "1s with moving average"
```

```
replot [0:1000] "3s data with gray moving average.txt" using "time:"
    →temperature" with lines title "3s with moving average"

replot [0:1000] "10s data with gray moving average.txt" using "time:"
    →temperature" with lines title "10s with moving average"
```

The plot command`was used with the following parameters:

- [0:1000] shows only the data belonging to the indices of the range
- "0.1s data with gray moving average.txt"`and the name of the file from which it takes the data, fomented in a certain way.
- using "time": "temperature" are the columns used for the X and Y axis of the graph, respectively
- with lines`and the parameter that linearly interpolates the graph samples
- title "0.1s with moving average" sets the title of the line

The replot server command to plot additional graphs in the same window.

```
#Frequency: 1000000000 Hz
#Gain: 30 dB
#Absolute time: Thu Oct 31 14:38:13 2024 UTC+2
#Integration period: 1 s
#Calibration period: 60 s
#LNB Equivalent Temperature: 120 K
#Offset: 13.8 dB
##############################################
 time power temperature LNBGain
#s    dBm   K           dB
##############################################
 102.47 -58.62 360.5 51.3
 102.48 -58.68 355.1 51.3
 102.48 -58.81 345.2 51.3
 102.48 -58.78 347.6 51.3
 102.48 -58.67 355.8 51.3
 102.48 -58.42 376.9 51.3
 102.48 -58.50 370.1 51.3
 102.48 -58.32 386.4 51.3
 102.48 -58.41 378.0 51.3
 102.48 -58.58 363.4 51.3
 102.48 -58.66 356.7 51.3
 102.48 -58.81 344.7 51.3
 102.48 -58.37 381.3 51.3
 102.48 -58.76 348.4 51.3
 102.48 -58.51 369.5 51.3
 102.48 -58.49 371.1 51.3
 102.48 -58.85 341.3 51.3
 102.48 -58.49 371.1 51.3
 102.48 -58.49 371.4 51.3
 102.54 -58.53 367.7 51.3
 102.54 -58.17 399.3 51.3
 102.54 -57.93 422.4 51.3
 102.54 -58.35 383.3 51.3
 102.54 -58.27 390.1 51.3
```

Figure 2.6: Example of data formatted to be readable on GNUPlot. All rows beginning with # are comments and columns are divided by a single space

In summary,`and a motorcycle software useful for the purpose of representing the data used in this thesis, and almost all the graphs obtained were derived with the help of GNUPlot.

2.6 Python

Python` and a high-level, object-oriented programming language with the main characteristic of being very flexible and easy to use, basing its syntax on the concept of well-ordered code, making it very readable unlike other languages. Other features of the language are[4]:

- variables are untyped, there` or means that variables are declared without specifying what type they are, since the computer takes care of interpreting it according to the type of value it contains.
- The language`is semicompiled, i.e.,`and is partially compiled into bytecode before being interpreted into machine language, this, together with the strong type control of variables and the garbage collector that dynamically frees unused memory, leads this language not to be particularly fast in execution time or mathematical computation, but leads it to be extremely flexible.

```
def add5(x):
   return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '    %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
       if ast[1].strip():
          print '= %s"];' % ast[1]
       else:
          print '"]'
     else:
        print '"];'
        children = []
        for n, child in enumerate(ast[1:]):
           children.append(dotwrite(child))
        print '    %s -> {' % nodename,
        for name in children:
           print '%s' % name,
```

Figure 2.7: Example script in Python. Important to note the absence of curly brackets to hold code and semicolons to terminate instructions, as they have been replaced by indentation and carriage [4]return, respectively

This language`was used for the main reason that GNU Radio`was programmed in Python, therefore, to create flexible blocks with respect to purposes, and`

been necessary to use it, and although it is not particularly fast, as mentioned earlier, it manages to handle real-time measurements with minimal delays, mainly brought on by the fact that we save real-time values to a text file formatted to be readable by GNUPlot.

Chapter 3

Methods used

3.1 SDR calibrations and characterizations.

This chapter lists the main methods used to characterize the SDRs used in order to make radiometric measurements and the runtime calibration, which is essential to avoid the drifts in temperature typical of these devices, during measurements.

3.1.1 Noise floor

Calibration by noise floor`was used to study the variation of the received signal when the SDR` is connected to an adapted 50Ω termination, to better understand its drift in temperature.

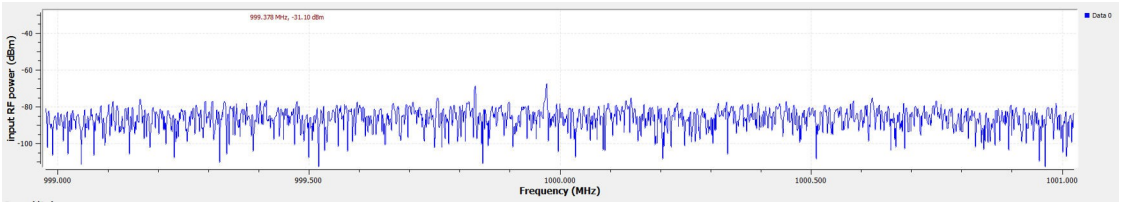

Figure 3.1: Example of 1GHz noise received as input from an adapted termination

By measuring the noise floor, ycannot` , of course, determine the dynamic range of the device, since you have to vary the input power to determine it, but you`and could exploit it to determine the variance of the saved values depending on the size of the moving average window that we do initially, and also on the integration time τ with which we average the input values. where we will later show that by increasing the integration time, the variance of the samples is reduced. Expected result knowing that moving window averages are in effect low-pass filters for the signal.

3.1.2 Signal generator

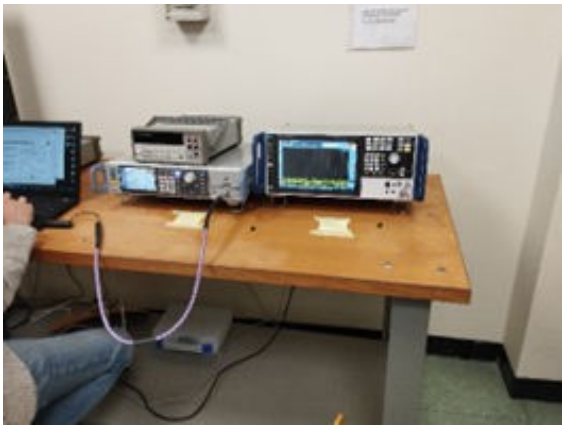

Figure 3.2: Measurement setup with signal generator

A signal generator`is an instrument with the task of being able to generate electrical signals of varying frequency, phase and amplitude by putting it out to a coaxial cable connector. It is mainly used to verify the operation of circuits or systems, or to check their outputs given a known signal, it connects to the circuit or device of interest through a 50Ω coaxial cable, i.e.`and of the same characteristic impedance as its connectors, to avoid impedance jumps that would cause reflections. In our case, the main purpose of the signal generator was to connect to the SDRs via a 50Ω coaxial cable to give input a 1GHz signal of varying frequency and amplitude (Between -95 dBm and -30 dBm), in such a way as define its dynamic ranges as the gain varies, since depending on how much the SDR amplifies, there are detectable input powers of different values.

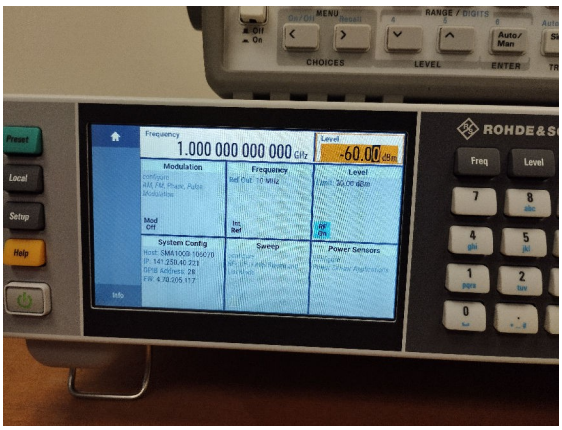

Figure 3.3: Example of generator setting

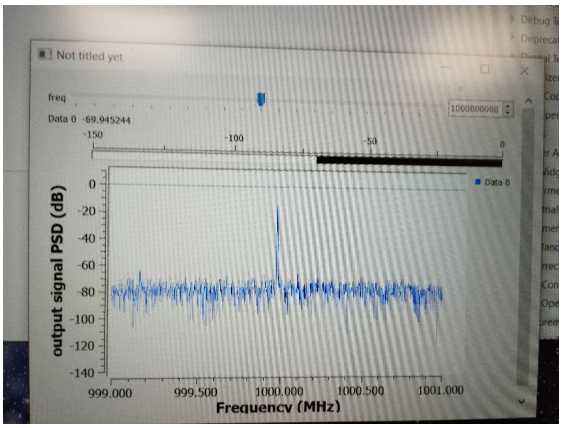

Figure 3.4: Example of 1GHz signal correctly detected by the device during characterizations

Calibration by signal generator`was useful in being able to calibrate the measured input power to the SDR against the input power, as well as bringing additional information about the by of the IF filter and the frequency shift (in ppm), as seen in 3.5.

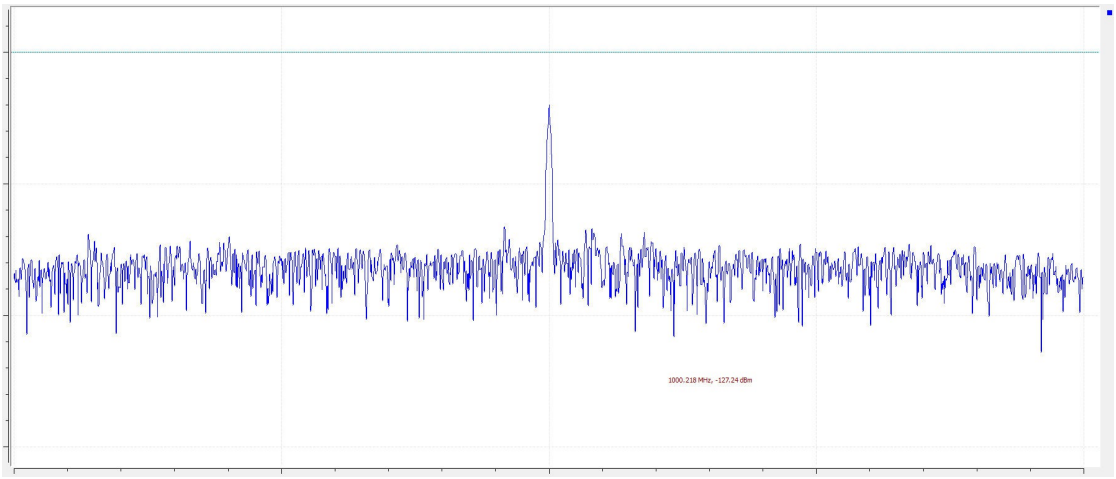

Figure 3.5: By generating a single sine wave with the generator (whose Fourier transform would be a pulse of infinitesimal width), one à determine the width of the IF filter experimentally

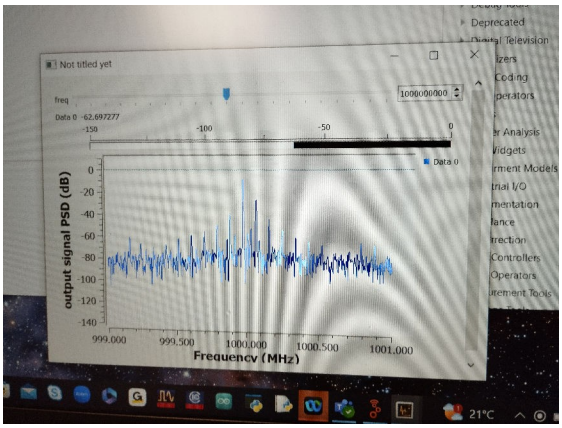

Figure 3.6: Example of saturation of the SDRs internal ADC

Interestingly, we get a frequency result about the saturation of the ADC, since we get spurious frequencies, i.e.`and unwanted frequency peaks, due to nonlinear distortion of the signal (clipping phenomenon)

3.1.3 Noise generator

The noise generator used`is a Noise Source HP 346C K01, which`is a device that, by exploiting the shot noise concept of a schottky diode, generates noise of a certain frequency (depending on the desired frequency) when voltages of 12/24V are applied. Shot noise (also called Schottky noise)`is a type of noise modeled by a Poisson process, derived from the fact that in reality`a the charges inside a diode are a discrete number. This type of noise`and independent of temperature, making it ideal for use as a noise generator, being able to become the dominant noise type with which to determine the noise figure of the devices under study, such as the SDR in our case.

The formula by which the noise figure of a device is determined by measuring the noise floor when the generator`is off and when`is on, e:`

$$= \frac{ENR}{Y-1} \tag{3.1}$$

Where Y`and the factor Y,
defined:

$$Y=\frac{P_{(on}}{Poff} \tag{3.2}$$

With P_{on} the power detected with the noise generator on, while P_{off} when the generator`and off. While ENR`and the Equivalent Noise Ratio defined in this way:

$$F=\frac{Thot}{Tcold} \tag{3.3}$$

Where T_{hot} is the hot temperature, while T_{cold} is usually defined as equal to T_0= 290K

With the HP 346C K01 noise generator we found determined the noise figure of the SDRs, which is useful for understanding the noise characteristics of the devices (Although Friis' formula the LNB gain determines the entire noise figure of the chain)

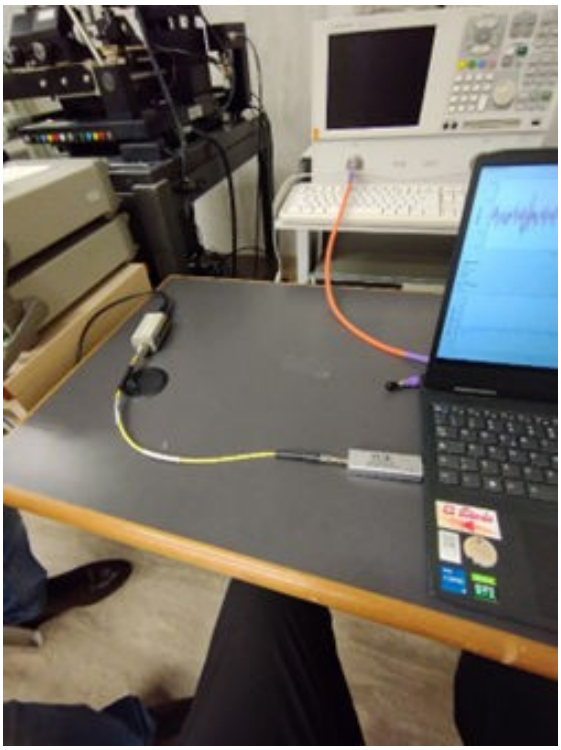

Figure 3.7: Measurement of noise with the generator

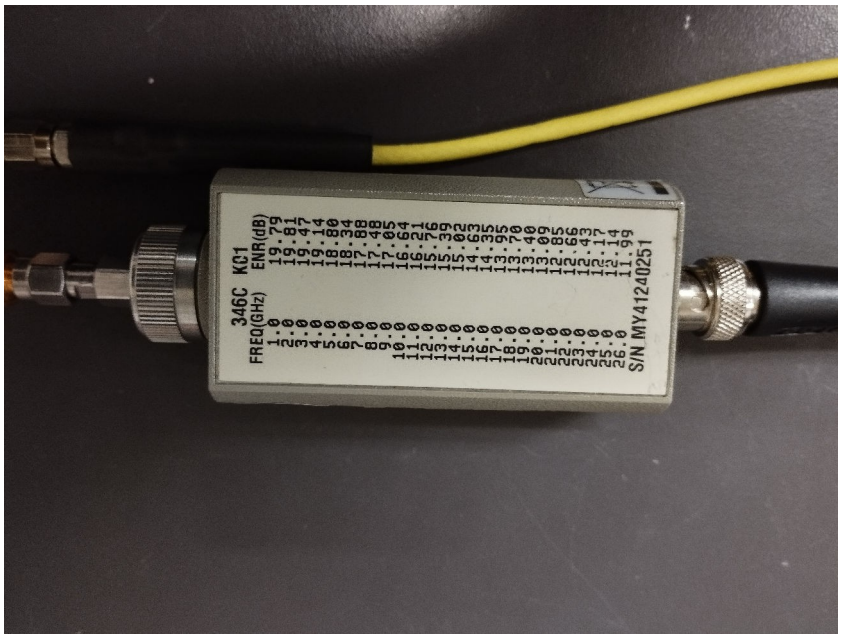

Figure 3.8: Noise generator, with noise figure table as frequency changes

3.1.4 LNB gain calibrations

During measurements, we need to periodically calibrate the system, based on the temperature detected by a blackbody, to calculate in runtime the effective gain of the LNB, assuming the equivalent noise temperature of 120K is known, the equations used to derive the formulas to be applied are:

Assuming the equivalent noise temperature of the LNB:

$$T_{RX} = 120K \tag{3.4}$$

And considering the equations related to the sensed power:

$$P_x = k_B(T_x + T_{RX})G_{LNB}B \tag{3.5}$$

Where P_x`and the input received power, k_B the Boltzmann constant, T_x the input equivalent noise temperature, G_{LNB} LNB gain and B the bandwidth. And the other equation:

$$P_{BB} = k_{(B)}(T_{BB} + T_{RX})G_{LNB}B \tag{3.6}$$

Where P_{BB}`and the power received as input from the black body and T_{BB} its temperature.

From which we derive:

$$G_{LNB}B = \frac{P_{BB}}{k_{(B)}(T_{BB} + T_{RX})} \tag{3.7}$$

Substituting 3.7 into 3.5, we get:

$$P_x = (T_x + T_{RX}) \frac{P_{BB}}{T_{BB} + T_{RX}} \tag{3.8}$$

Solving *for* T_x we get:

$$\frac{T_x + T_{RX}}{T_{BB} + T_{RX}} = \frac{P_{(x}}{P_{BB}} \tag{3.9}$$

Where $\frac{P_x}{P_{BB}}$ is also called the Y factor. :

$$T_x + T_{RX} = \frac{P_{(x}}{P_{BB}} (T_{BB} + T_{RX}) \tag{3.10}$$

And finally:

$$T_x = (T_{BB} + T_{RX}) \frac{P_{(x}}{P_{BB}} - T_{RX} \tag{3.11}$$

Whose 3.11`and the formula that`has been implemented within the algorithm to calculate the equivalent sun noise temperature at runtime, using a black absorbing panel as the reference black body, which is placed in front of the dish illuminator during the calibration phase, where the power P_{BB} is measured over a 60-second inter- vallation. The said power is subsequently averaged and used to determine its Y factor to be used within the formula, along with the black body temperature T_{BB}, which is entered manually at the end of calibration.

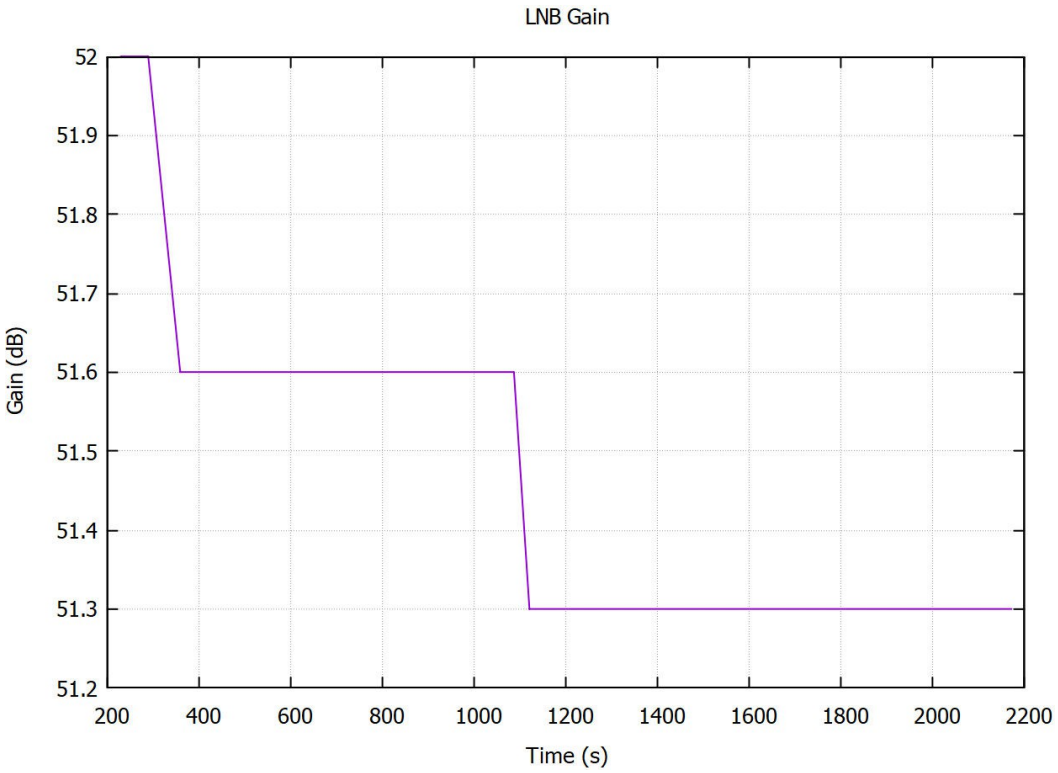

Figure 3.9: Variation of LNB Gain obtained through runtime calibration In Figure 3.9 we can see an example of measurement with multiple calibrations performed at runtime, leading G_{LNB} to take the most accurate value for the LNB, starting from an arbitrary default value of 52 dB and ending toward 51.3 dB.

Chapter 4

GNU Radio scripts

4.1 Block diagram

Using GNU Radio, we define via software all the processing of the signals received from the SDR in the frequency of interest, through a block diagram that manages the signal flow.

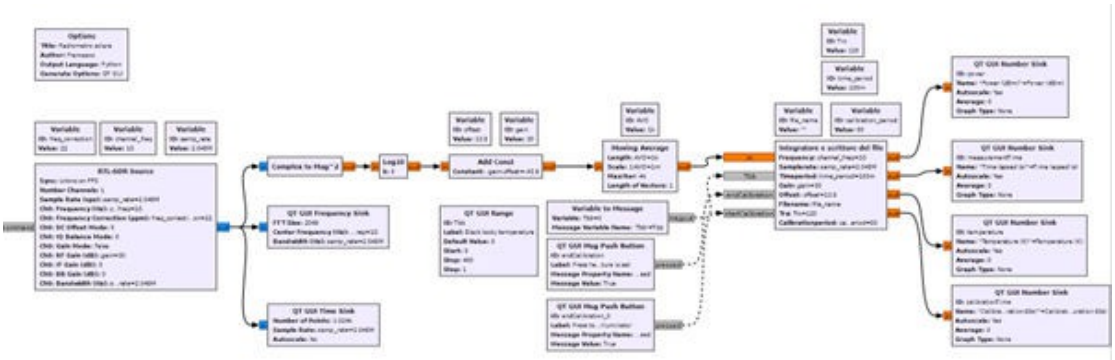

Figure 4.1: GNU Radio script block diagram

Essential block for communicating with the SDR, i.e.,` and setting the tuning frequency and other parameters to receive the signal of interest,`and the RTL-SDR Source, a block specifically dedicated for the use of an RTL-SDR, as seen in Figure 4.2.

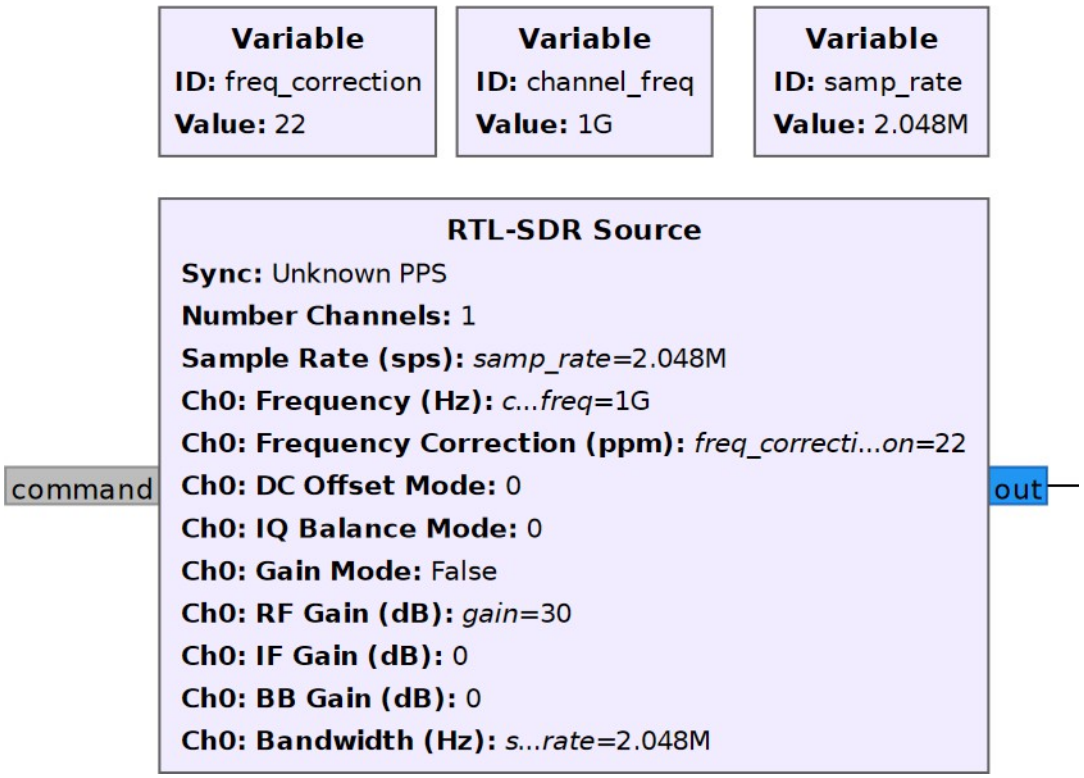

Figure 4.2: Source block with which we communicate with the SDR

As shown in Figure 4.3, after receiving the signal, we derive its square modulus and convert it to dBm, after applying the offset due to the gain of the SDR and the intrinsic offset of the receiver.

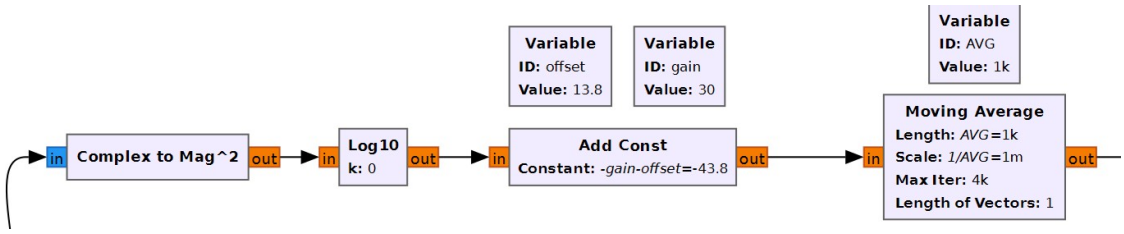

Figure 4.3: Block sequence deriving signal power

Finally, it is made the moving average of N samples, to reduce the variance of the samples received as input, making them mò accurate and, most importantly, reducing the number of samples to be saved to file, since we save about 2 million per second and they do not carry a large enough in- formation to have to save them raw, making it useful to use this technique, together with the integration over time of M samples to which the arithmetic mean is then made.

4.2 Python block

GNU Radio's python block is used as a module to implement non-pre-existing algorithms, as in this case, where we`and had to implement an integrator to average N samples over a time τ and be able to write in a formatted manner to a text file, as well as implement the calibration algorithm based on black body only.

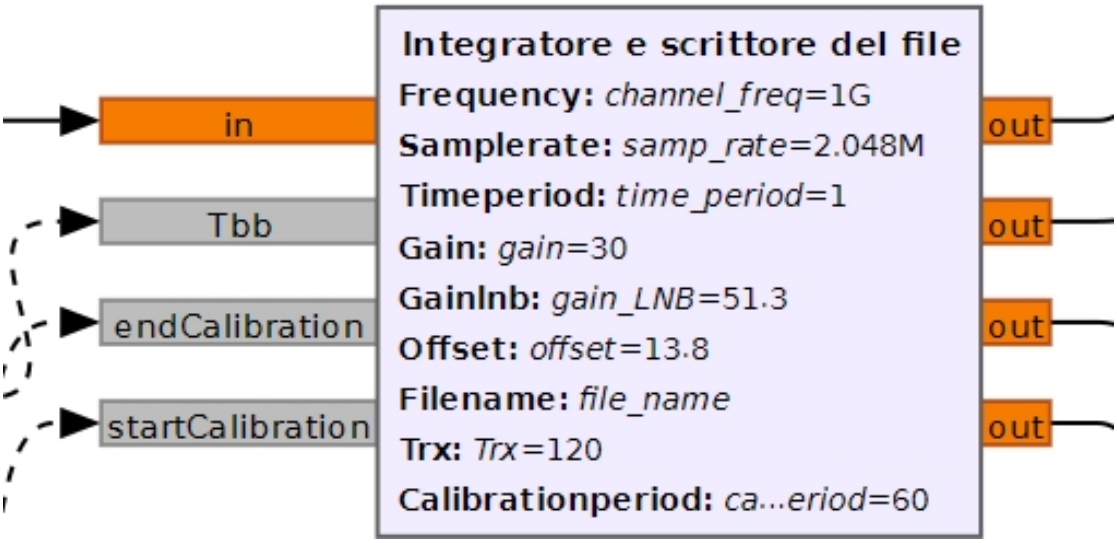

Figure 4.4: Python block within GNU Radio.

The following will describe the most important parts of the Python code (described completely in A.1):

```
gr.sync_block.__init__(
        self,
        name='Integrator and file writer', # will show up in
            →GRC
        in_sig=[np.float32],
        out_sig=[np.float32, np.float32, np.float32]
    )
```

This section of code`and the object constructor that describes the behavior of the GNU Radio python block of type sync, that is`and based on the signal input event within the block. As you can see, in addition to defining the name of the block, you define in sig and out sig, which are the inputs and outputs of the block, respectively. These variables are vectors that accept as elements the type of value that is expected, i.e.`and, in our case, the float32 type, which means that we expect numeric values represented in floating point at 32 bits in size.

```
if (time.time()-self.calibrationTime)<self.calibrationPeriod:
        self.tempPbb =(pow(10,((input_items[0][0]-30)/10)))
        self.Pbb += self.tempPbb
        self.temperature = self.tempPbb/(self.kb*self.
            →bandwidth*(pow(10,((self.gainLNB)/10)))) string="
        "  +  '{:.2f}'.format(round(time.time()-self.
            →absTime, 2))+ " "+ '{:.2f}'.format(round(10*
            →math.log(self.tempPbb,10)+30, 2))+ " "+ '{:.1f
            '→ }'.format(round(self.temperature,  1))+  " "  +
            → '{:.1f}'.format(round(self.gainLNB, 1))+ "\n" with
        self.path.open("a") as f:
            f.write(string)
        self.i += 1
```

In this block we show the heart of the script's behavior during calibration, i.e.`and the integration of the power detected during 60s, converted to linear from dBm, to make the average to be used as the P_{BB} value in the temperature calculations of the values measured later. Of the values measured during calibration time, one also makes a conversion to temperature to write it into the temperature column in the text file by exploiting the formula:

$$T_{BB} = \frac{P_{BB}}{kBGLNB} \quad (4.1)$$

Which is easily obtained from the thermal noise formula:

$$P_{BB} = k_B G_{LNB} T_{BB} B \quad (4.2)$$

Within the code you à also see the formatting of the string with the values in time, power and temperature to be written within the file, an operation that have- determined experimentally take a maximum of 10ms computation time, time that is negligible compared to the data acquisition times we usually deal with on the device.

```
if self.flagTbbSet:
        self.Tbb=self.tempTbb
        self.gainLNB= 10*math.log((self.AvgPbb / (self.kb*(self.
            →Tbb+self.Trx)*self.bandwidth)), 10)
        self.flagTbbSet= False
```

Here we show the section by which we calculate T_{BB} and G_{LNB}, which will be used until the next calibration. It is located within an if structure dependent on a

flag perch and operation that only has to do once per calibration, moreover, the formula by which G_{LNB} is calculated is follows:

$$G_{LNB} = 10 \log_{10}(k_B(T_{BB} + T_{RX})B) \tag{4.3}$$

$10 \log_{10}(...)$ is done to convert it to dB, a unit with which gain is usually represented.

```
if self.flagMeasurement and (time.time()-self.time)>= self.period:
        self.power= self.power / self.i
        Y= self.power/self.AvgPbb #Y factor self.temperature    =
        ((self.Tbb+self.Trx)*(Y))-self.Trx
        #self.temperature   =   (pow(10,((self.power-30)/10)))/(self.
            →kb*self.bandwidth)
        #Prepare string and write to file.
        string=" "    +   '{:.2f}'.format(round(self.time-self.absTime,
            '→  2)) " "  +      +   '{:.2f}'.format(round(10*math.log(self
            →.power, 10)+30, 2))+ " "+ '{:.3f}'.format(self.
            →temperature)++'{:.1f}'.format(round(self.
            →gainLNB, 1))+ "\n"
        with self.path.open("a") as f:
            f.write(string)
```

Here is the most important section, where we average the samples in power over the integration time τ, calculate their temperature using formula 3.11, which and usable, having calculated P_{BB} (AvgPbb in the code). Immediately thereafter, as in the calibration phase, we prepare the string with the values in power and temperature, as well as the time in which the sample was calculated and the current gain of the LNB.

```
self.startCalibrationPort= 'startCalibration'
    self.message_port_register_in(pmt.intern(self.
        '→  startCalibrationPort))
    self.set_msg_handler(pmt.intern(self.startCalibrationPort),   self.
        →handle_msgStartCalibration)
    self.startCalibration= False
```

With this fragment we describe another particular features of GNU Radio, namely and messages, which are packets that can be sent through dedicated innput and output ports consisting of an identifying key and a value, useful for sending control commands to blocks that support it, or for, as in this case, allowing interaction between GUI elements and variables within the Python block. Specifically, with this fragment we create the GNU Radio port that allows a Boolean value to be taken

sent by the calibration start button, with all the instructions for connecting the port to the function (via the handler) that handles its received packet.

```
def handle_msgStartCalibration(self, msg):
    if self.calibrationTime != 0 and not self.startCalibration:
        self.endCalibration= False
        self.calibrationTime= 0
    self.startCalibration= msg
```

Finally, with this second fragment we continue the example by describing the function that is called by the handler implemented earlier, Interestingly, it is necessary to always put the received message as a fomral parameter, since`it is of interest to us to handle it and, in this specific fragmentyà see how we assign it to the startCalibration flag, while setting 2 others to False and 0 respectively for the purpose of being able to re-run the calibration sequence.

4.3 Graphical user interface (GUI)

The graphical user interface (called GUI, Graphical User Interface)`was made by taking advantage of GNU Radio's native QT blocks (Graphical Library for creating windows with various elements, including numbers, plots and buttons) in order to nimbly create a graphical user interface useful for runtime measurement purposes, I note that all data are saved to a file that`e can observe and manage in post-measurement with GNUPlot.

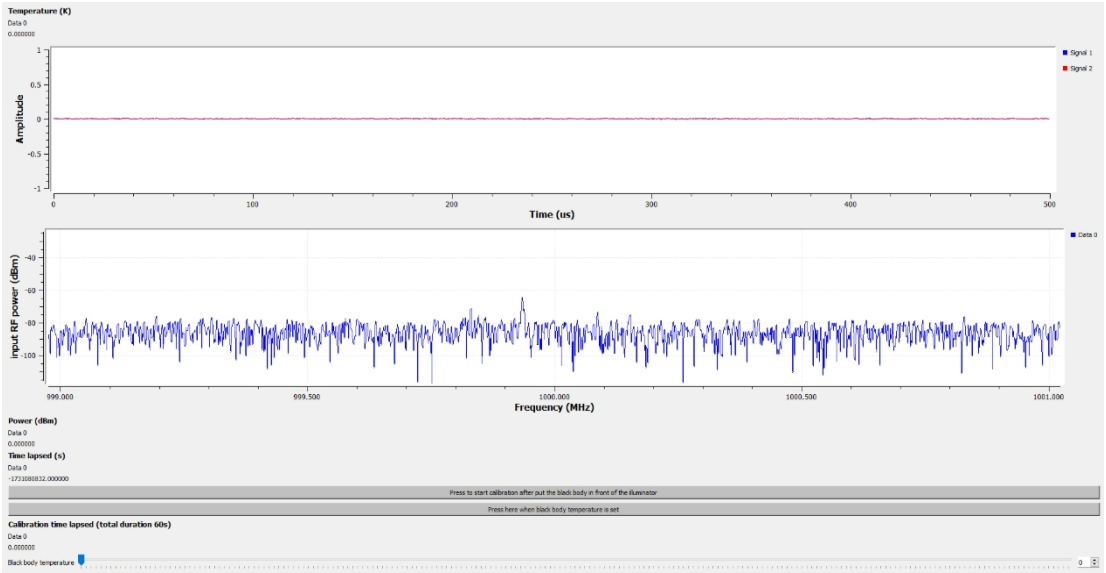

Figure 4.5: Graphical interface designed to display the values measured by the instrument and to calibrate it at runtime

The interface`consists of ìbsections:

- Time sink and frequency sink: blocks showing the signal over time and its FFT
- Number sink: blocks showing the number received as input via GNU Radio
- Calibration section, which allows LNB gain calibration to be performed at runtime

4.3.1 Time sink and frequency sink

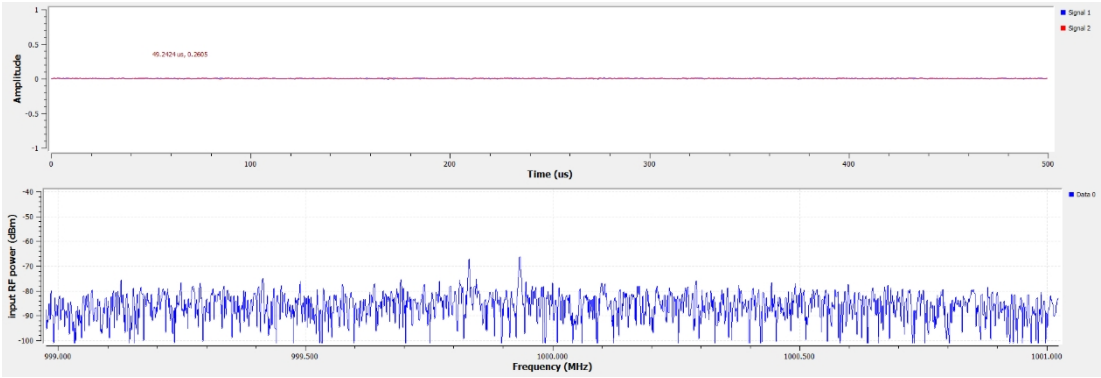

Figure 4.6: Section showing the signal in time and frequency

This section allows, with the time sink, to observe the last 1024 I/Q values received in real time, to oserve how much the signal varies and to prevent saturation of the ADC in case the signal is too amplified. Through the frequency sink we ca observe the discrete Fourier transform (DFT) of the signal in real time, using an FFT on 2048 samples, this block facilitates the visualization of the received frequencies, allowing us to see if there are accidentally received spurious signals (we just want to receive noise, which`is band uniform by definition) or to observe if the frequency correction applied on the SDR is correct. Thanks to these blocks, we have better control of what`or we receive and, most importantly, it already allows`a to get the receiving cir- cuitry up and running, taking away transients due to temperature drifts in a matter of minutes before making the actual measurements, making them mo accurate

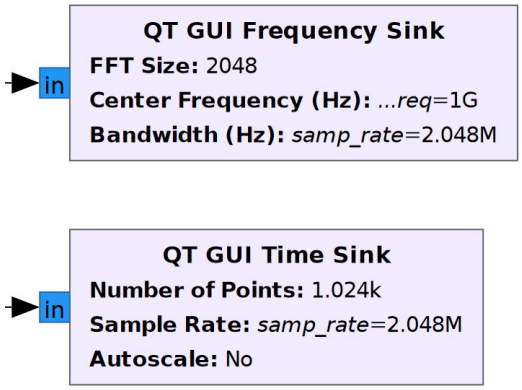

Figure 4.7: GNU Radio blocks showing the signal in time and frequency, they want a complex numbero as input since they receive both in-phase and quadrature phase signal (I/Q components)

4.3.2 Number sinks

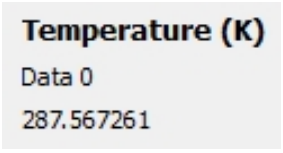

Figure 4.8: Number sink showing temperature in Kelvin

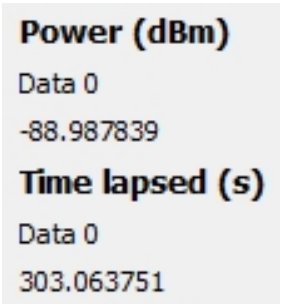

Figure 4.9: Number sinks showing input power (in dBm) and time elapsed since the start of measurements

These blocks allow reading the power input values to the SDR and the calculated tem- perature through the formula, derived from the calibration concepts with the black body only (assuming known $T_{(RX))}$:

$$T_X = (T_{BB} + T_{RX}) \frac{P_{(X}}{P_{BB}} - T_{RX} \tag{4.4}$$

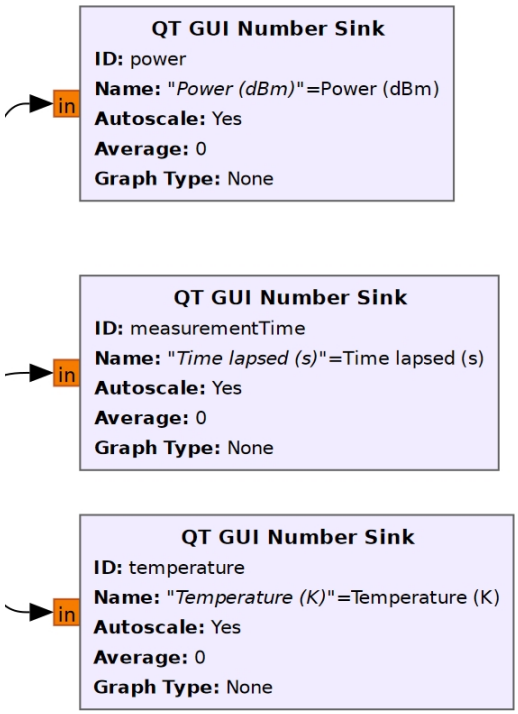

Figure 4.10: QT blocks obtruding real-time input values on the GUI. The blocks were set to receive values in float32 (real 32-bit floating-point numbers) with autoscale enabled and a refresh time of 0.1 seconds, low enough to show the values in real time, without varying too quickly for the human eye

4.3.3 Calibration section

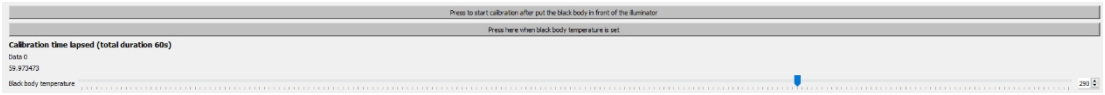

Figure 4.11: Section of the GUI that allows you to do runtime calibration.

How to use runtime calibration

To start making measurements, you must first calibrate the si- syste- m at least once to determine the current gain of the LNB G_{LNB}, and then you can`or redo the calibration whenever you feel it is necessary. The steps to follow through the GUI are as follows:

- Start the GNU Radio script to begin measurements
- Wait about 2-3 minutes to clear the drift in temperature of the transient SDR
- Press on the button "Press to start calibration after put the black body in front of the illuminator" as in 4.12 after placing the desired black body (like a

absorbing panel of the type of 4.15) in front of the dish illuminator (LNB) and wait

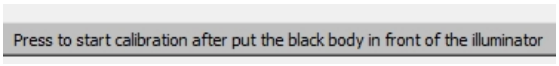

Figure 4.12: Calibration start button

- After waiting a time equal to 60s (unless otherwise specified in the parameters), where you can see the elapsed time in the number sink titled "Calibration time lapsed (total duration 60s)," as in 4.13, you remove the black body from in front of the illuminator (after the calibration time has passed it stops integrating input power) and, after setting the black body temperature with the appropriate input range titled "Black body temperature" as seen in 4.13, you press on the "Press here when black body temperature is set" button as in 4.14

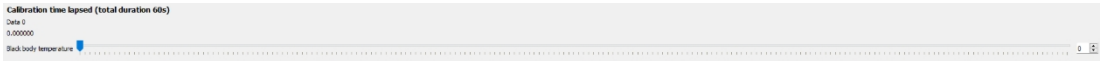

Figure 4.13: Section of setting the black body temperature and the time elapsed during calibration

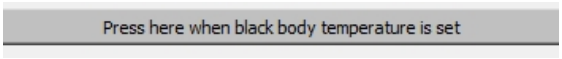

Figure 4.14: Calibration end button that initiates the actual measurements

- After taking these steps, the calibration has been done and from now , until we do a calibration or close the script, it will be to integrate the input power and save it to a file, the name of which is automatically- generated based on the timestamp or set by us among the parameters of the GNU Radio project

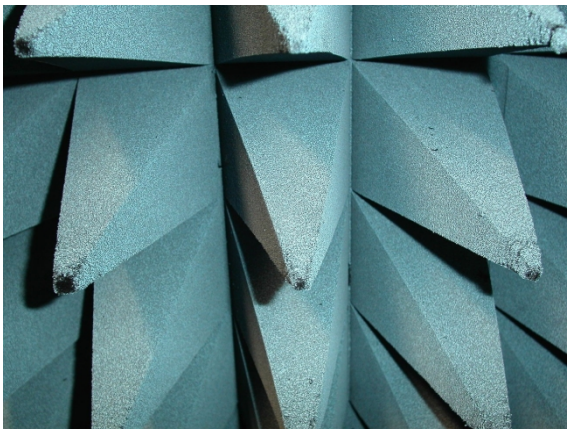

Figure 4.15: Example of a black body using a panel made of RAM (Radiation Absorbing Material), with a pyramidal geometry in such a way as to be as absorbent as possible, and thus avoid impedance mismatch with the preceding medium (air)[5]

Chapter 5

Results obtained during the characterization of SDRs

During preliminary tests, we`and tried to characterize two RTL-SDRs (one black and the other gray) to determine which was the best in terms of accuracy, precision, by in temperature and the like. Methods were used previously mentioned to characterize them and respectively you è , as already`a said:

- signal generator to determine inherent SDR offset to make the measured powers accurate.
- The noise generator to determine its noise figure, which determines its noise`a in measurement, although easily solved by adding an LNA (Low Noise Amplifier) at the beginning of the chain, which in our case`and implemented using the LNB.
- The noise floor measurement`was used to determine its drifts in temperature and the variance of the noise it measures as integration times vary or from the use of the moving average filter.

Using the methods listed in Chapter 3, it was possible to characterize both SDRs in various aspects useful in determining accuracy of the measurements.

5.1 Dynamic range as gain changes

The dynamic range, called dynamic range in Italian,`and the ratio between the maximum value and the minimum value receivable by an ADC, which determines, in our case, for what signal voltages (and therefore what powers) are manageable by it and interpretable into unique values, digitized for use by the pc.

The determination of the dynamic range and the determination of its maximum and minimum values as the gain changes`was determined through the use of the signal generator.

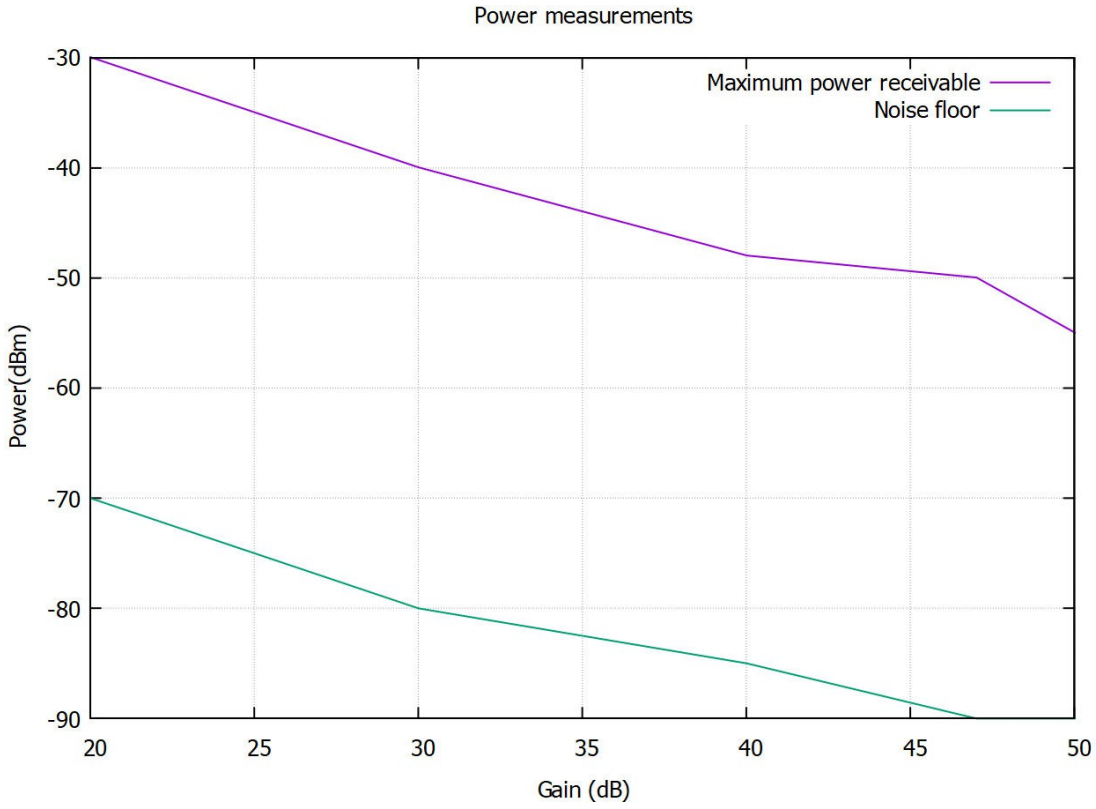

Figure 5.1: Dynamic range of gray SDR as gain changes

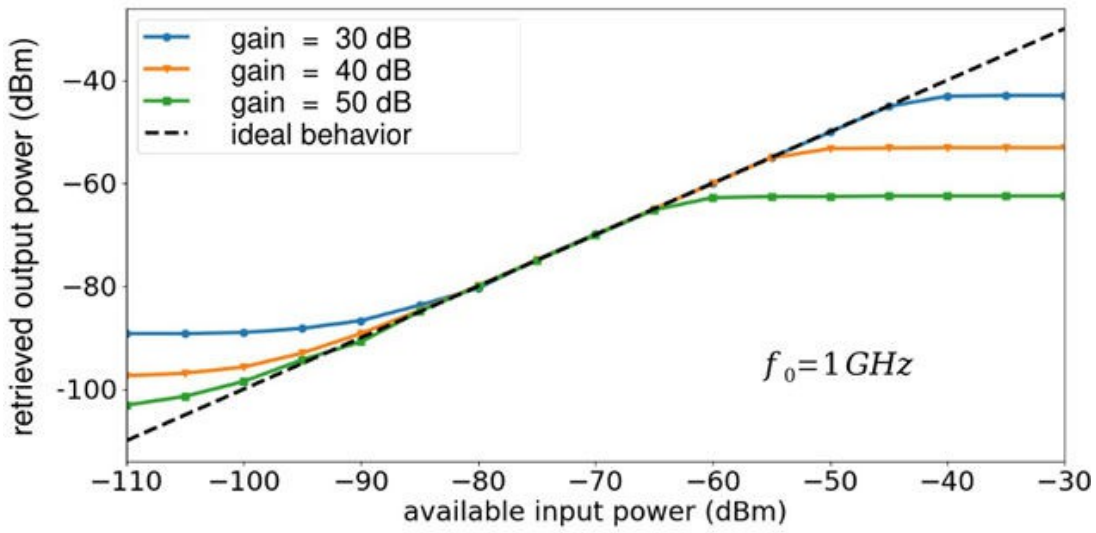

Figure 5.2: Dynamic range of black SDR as gain changes[13]

As you observe the dynamic ranges of the two SDRs remain constant at about 40dBm and 60dBm respectively, but shifting downward as we increase the gain, since if we increase the gain of the SDR, the ADC saturates with lower powers.

Gain(dB)	Offset(dB)	Pmax(dBm)	Noise floor(dBm)
50	0.5	-55	-90
47	2.7	-50	-90
40	5.4	-48	-85
30	7.8	-40	-80
20	6.5	-30	-70

Table 5.1: Experimentally derived calibration values of gray SDR

Gain(dB)	Offset(dB)	Pmax(dBm)	Noise floor(dBm)
50	13.8	-65	-95
40	14.1	-55	-85
30	13.8	-45	-80

Table 5.2: Experimentally derived calibration values of black SDR In addition to

determining the dynamic range and its limits as gain changes, we
and could define an offset (in dB) by which the measured values are made accurate, known the power of the input signal, the value of which also varies with gain.

5.2 Variance according to τ

Another important feature and the variance of the samples received as input, which and dependent on the integration times implemented in the linear average of the above measured samples.

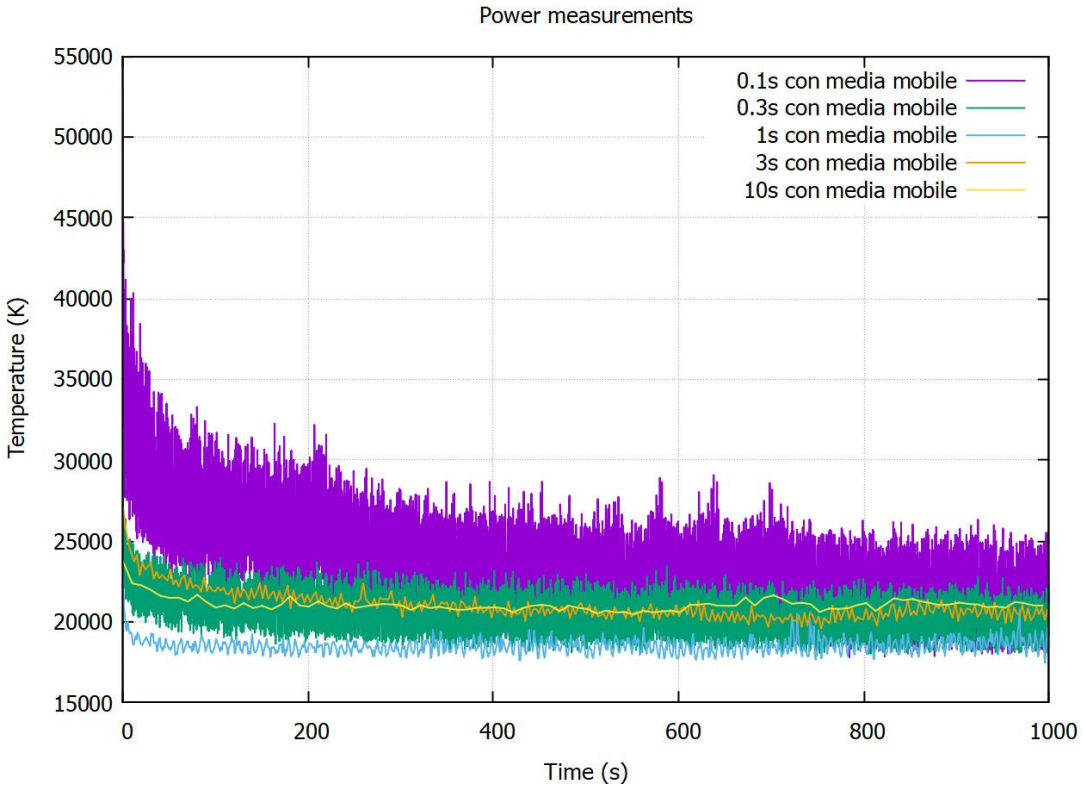

Figure 5.3: Measurement realizations based on the integration times τ set Maintaining a

constant moving-window average (useful as an initial filtering of the samples received) one à observe how, after an initial transient due to the drift in temperature of the instrument, all realizations of the measurement lie around the same mean value, with the only difference being`due to the variance of the samples, which decreases with increasing integration times, as expected from theory.

The variance formula used to graph the fluctuations in 5.4 e:`

$$\sigma_{T(eq} = \frac{TRX}{\sqrt{\tau B}} \quad (5.1)$$

With T_{RX} the equivalent system noise temperature, τ integration time used for arithmetic averaging, and B the bandwidth of the SDR.

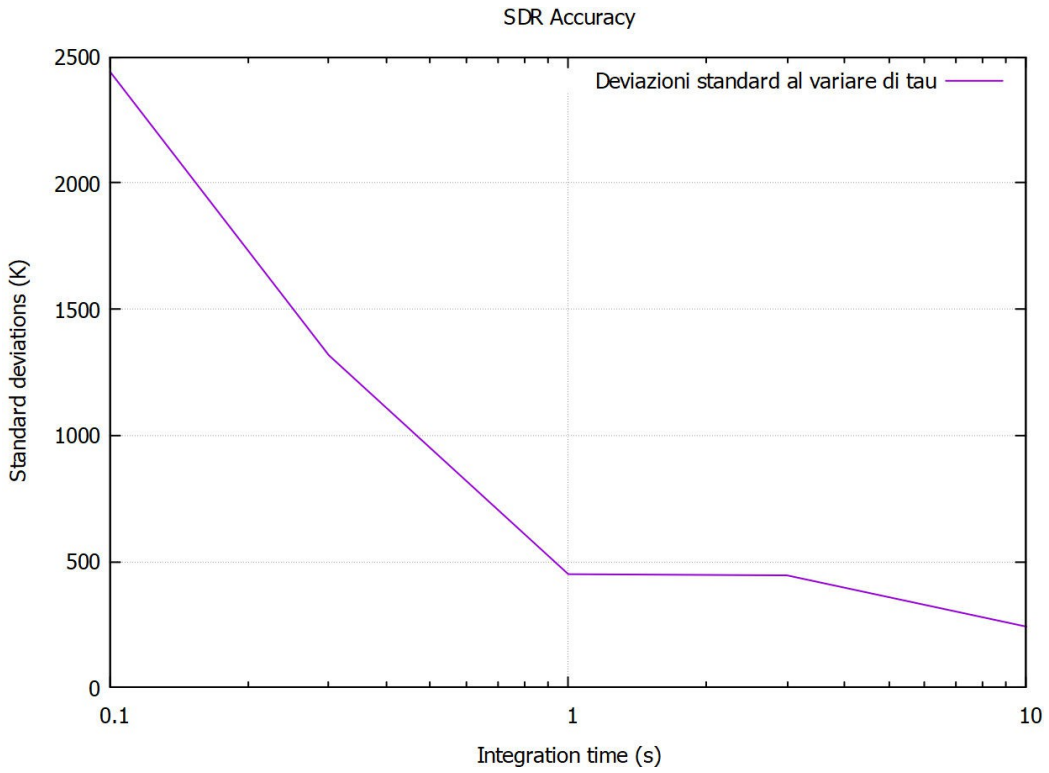

Figure 5.4: Graph of measurement variance according to integration time τ of the gray SDR

Another important factor is that, as seen in 5.4, the variance at a certain stops going down as τ increases, indeed, it even begin to go up (as seen in 5.5), making the measurements even more uncertain, leading to the need to find a suitable τ value to avoid too much fluctuation in temperature.

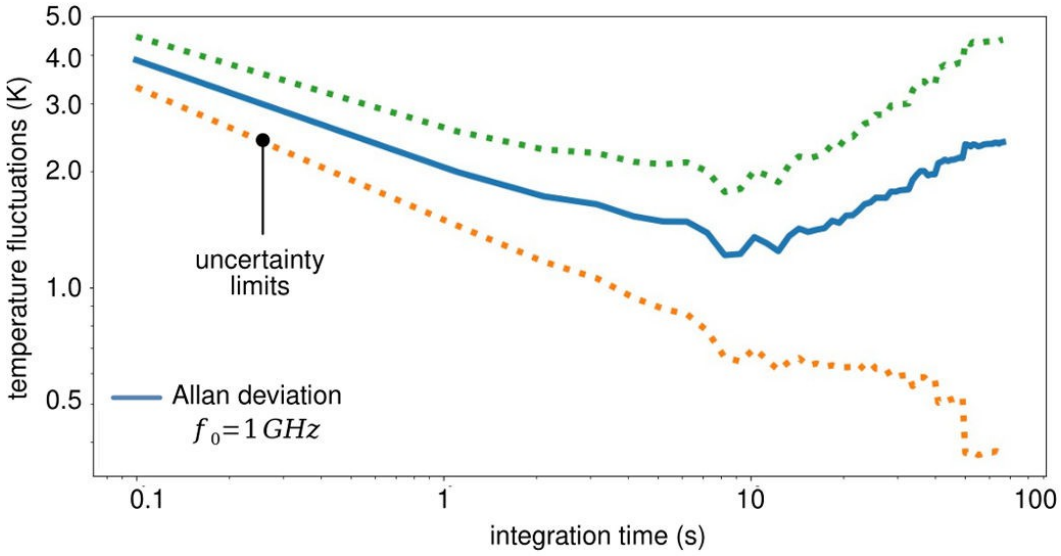

Figure 5.5: Allan variance as black SDR integration times change

5.3 Noise figure

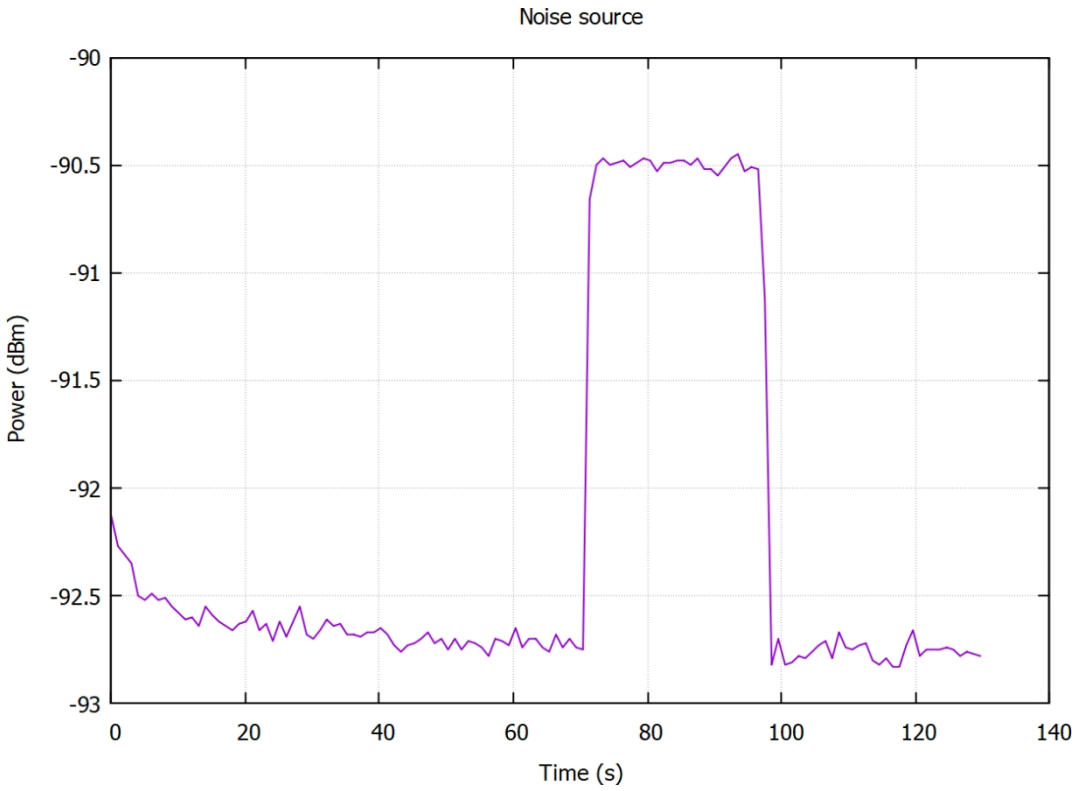

Figure 5.6: Measurements taken while using the noise generator 1GHz, 2MHz bandwidth and 50dB gain, where the peak`and when it was on, the rest of the mat, however, when it was off

The determination of the noise figure`was done through the use, as we`e already`a mentioned in 3.1, of a 1GHz noise source and by measuring the change in the noise mat detected by the SDR when the source`is off and when`is on. We know that the HP 346C K0 source *has an* $ENR_{dB}$= 19.79 dB at the frequency of 1GHz, so`o, converting it to linear, we can use formula 3.1 to derive the noise figure of our device based on the values of 5.6 when in the gray SDR, obtaining:

$$ENR_{lin} = {}^{10(ENR)(dB)/10} = 95.28 \tag{5.2}$$

Calculating the Y factor in logarithmic form:

$$Y_{dB} = P_{on} - P_{off} = -(90.5 - 92.7) = 2.2dB \tag{5.3}$$

Converting it to linear:

$$Y_{lin} = {}^{10Y_{dB}/10} = 1.66 \tag{5.4}$$

We can calculate the noise figure in linear form, obtaining:

$$F_{lin} = \frac{(ENR)_{(lin)}}{Y_{lin} - 1} = \frac{95.28}{1.66 - 1} = 144.36 \tag{5.5}$$

Which in logarithmic form equivalent to:

$$F_{dB} = 10 \log_{10} F_{lin} = 21.6dB \tag{5.6}$$

Therefore`o we can see that these devices can be particularly noisy, making it necessary to use an upstream LNA in order to make accurate measurements.

gain settings (dB)	G_{sdr} (dB)	F_{sdr} (dB)
30	44.3	22.3
35	49.4	18.4
40	54.4	13.1
45	58.8	10.8
50	63.6	7.0

$f_0 = 1\,\mathrm{GHz},\ B_{sdr} = 2\,\mathrm{MHz}$

Figure 5.7: Characterizations of gain and noise figure of the black[13] SDRAs à be

seen in Table 5.7 derived with the characterizations of the SDR ne-
ra, these values always depend on the gain set and, the hardware à be
particularly different, leading to completely different characteristics, although, when using an LNA , to avoid saturation of the ADC, one must necessarily re- durate the gain of the SDR, leading to a higher` noise figure that is for`or minimized thanks to the LNA itself.

Chapter 6

Experiment

6.1 Preliminary phase: calibrations and testing

Initially, before making the actual measurements with the sun, we`and used the system in a simplified environment, where we`and placed the dish, connected through a 50Ω coaxial cable, toward the open window, to see the differences in detection between the black body (an absorbing panel) and the sky we detect from the window, thus allowing us to test the script while waiting for the right time allowing us to make improvements until we get there`or what we expected. The SDR eventually chosen`and the black one, having proven to be mè consonant for noise measurement applications.

Figure 6.1: Setup initially set up to do the preliminary tests from the window

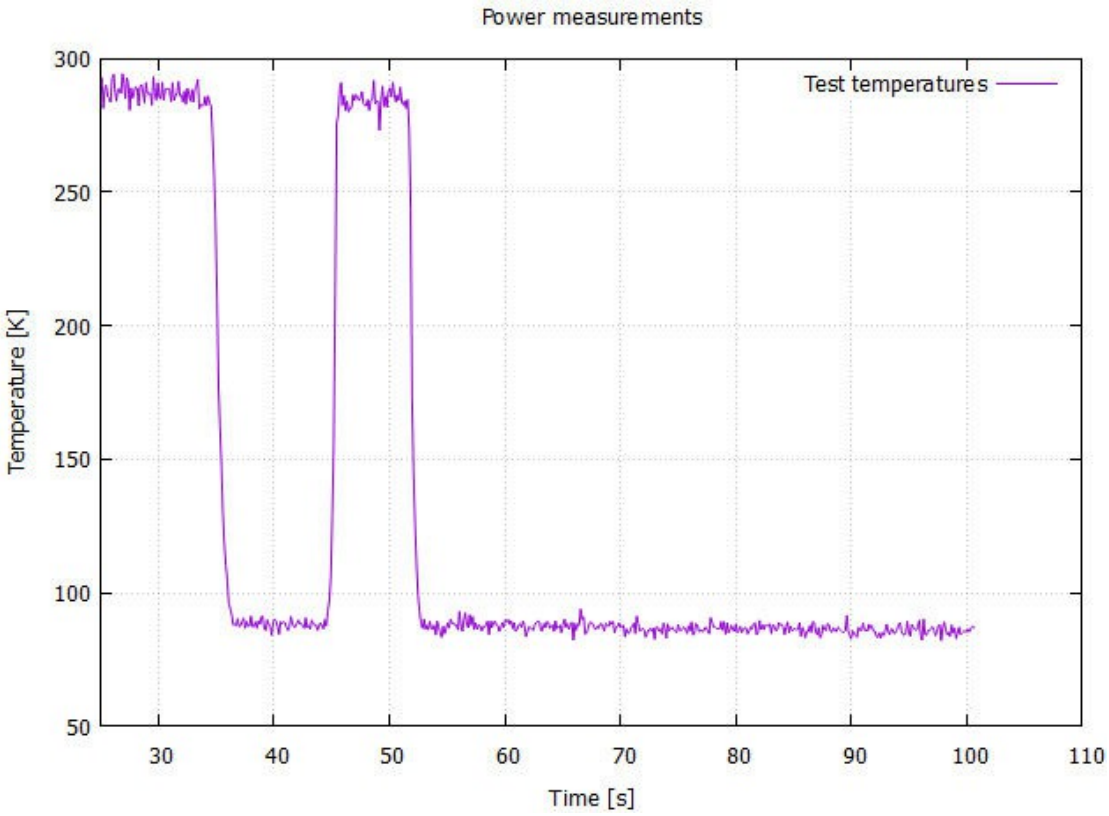

Figure 6.2: Graph of the preliminary system test, where we`and tried to measure the temperature of the sky and of a black body placed by hand in front of the illuminator

Aì see in 6.2 the radiometer, after preliminary calibrations of the LNB gua- dagno, manages to correctly detect the blackbody (on 290K) and the sky it detects from the open window, which is on 60-70K (Figure 6.2), since it was very proba- bly taking part of the window itself as well, not having fully exposed the dish to the outside and having set it on 50° elevation.

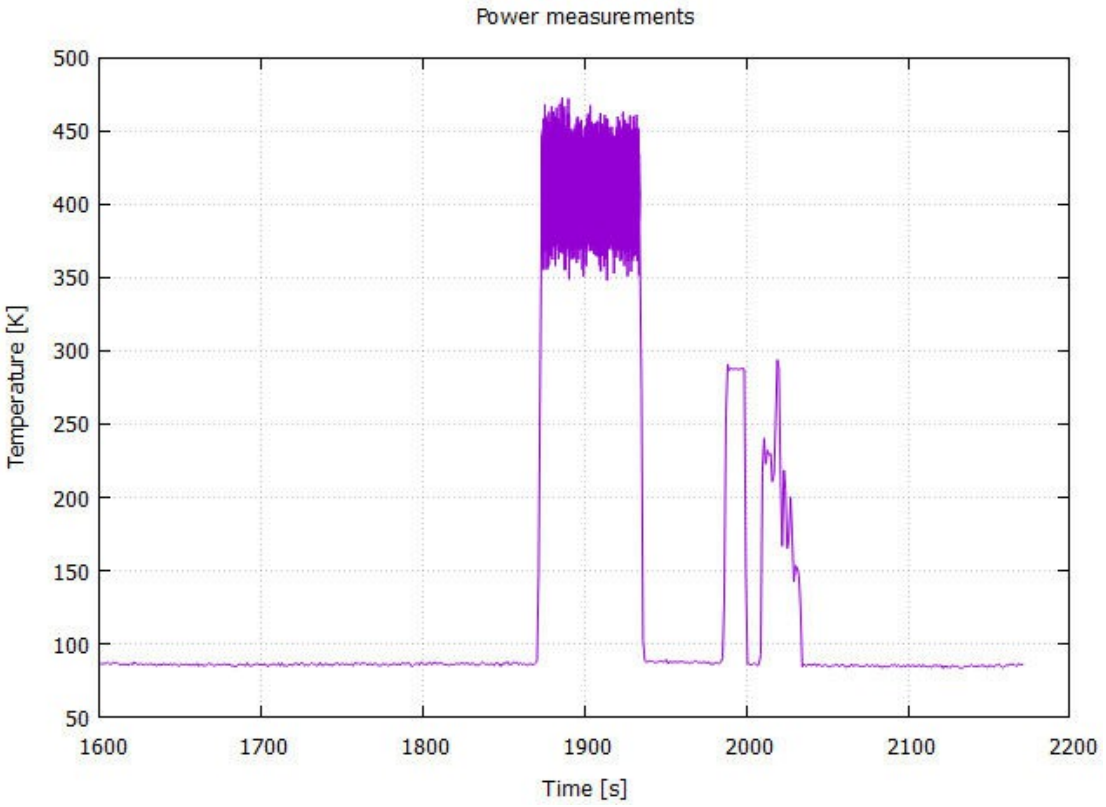

Figure 6.3: Example of script execution with runtime calibration and interposition of obstacles of various temperatures between illuminator and dish

Figure 6.3 shows an example of calibration with the same setup as the mi- surations in 6.2, where we first calibrate the system using the blackbody at room temperature (about 290K) and then try to see if it detects it correctly. As can be seen on the peak obtained just before 2000s, It consistently detects the absorbing panel temperature at about 280-290K. The variances obtained thereafter are ten- tions with the hand to have its temperature detected by varying the distance of it from the illuminator, as you see the temperature detected and you get the mo truthful one when`and particularly close, which` and the same distance assumed with the absorbing panel during calibrations.

6.2 Measurement the sun's electromagnetic emission at the frequency 10.70GHz

Figure 6.4: Photos of the system used during the experiment on 31-10-2024

On Oct. 31, 2024`the first experiment was carried out to demonstrate the feasibility`a of making microwave measurements of the sun with the described system.

The parameters set in the instrument are as follows:

- Sampling frequency: 2,048 MS/s
- Center frequency: 1GHz
- Bandwidth: 2,048 MHz
- Frequency correction: 22ppm (parts per million)

- Offset: 13.8 dB
- SDR gain: 30 dB, set not to maximum (50dB) to avoid saturation of the ADC due to the 51dB of the LNB, although it Noisy the device, makes n negligible being the second stage of the chain
- Moving average window size: 1000 samples
- Integration time τ : 1s
- Equivalent noise temperature LNB T_{RX}: 120K
- Calibration time: 60s

Initially we` and calibrated the system using a black body as the calibration temperature, which stood at about the temperature of 296K, based on the temperature of the surrounding environment.

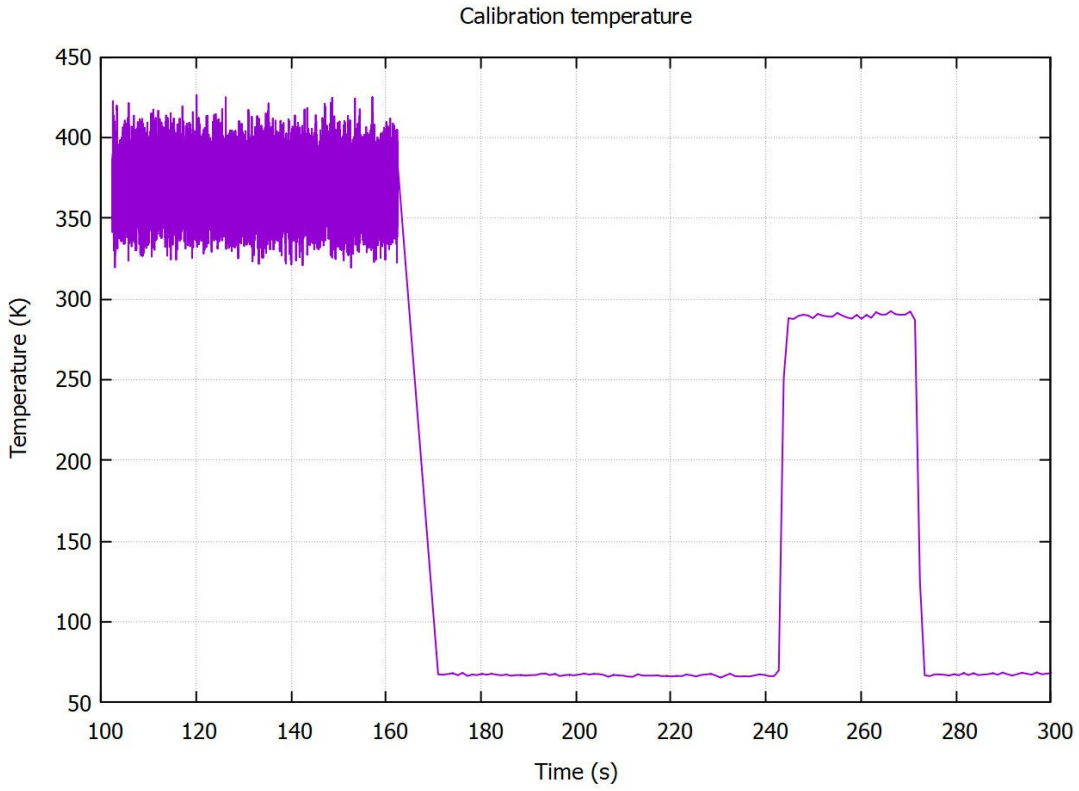

Figure 6.5: Calibration by measuring the emission of a blackbody

As shown in the figure , from 100 to 160 seconds we`and carried out the calibration of the system, averaging the raw data in detected power to then calculate the LNB gain, note the temperature of the blackbody. The detected temperature of about 290K between 240 and 280 seconds`and the temperature detected by the blackbody itself to make sure it was well calibrated.

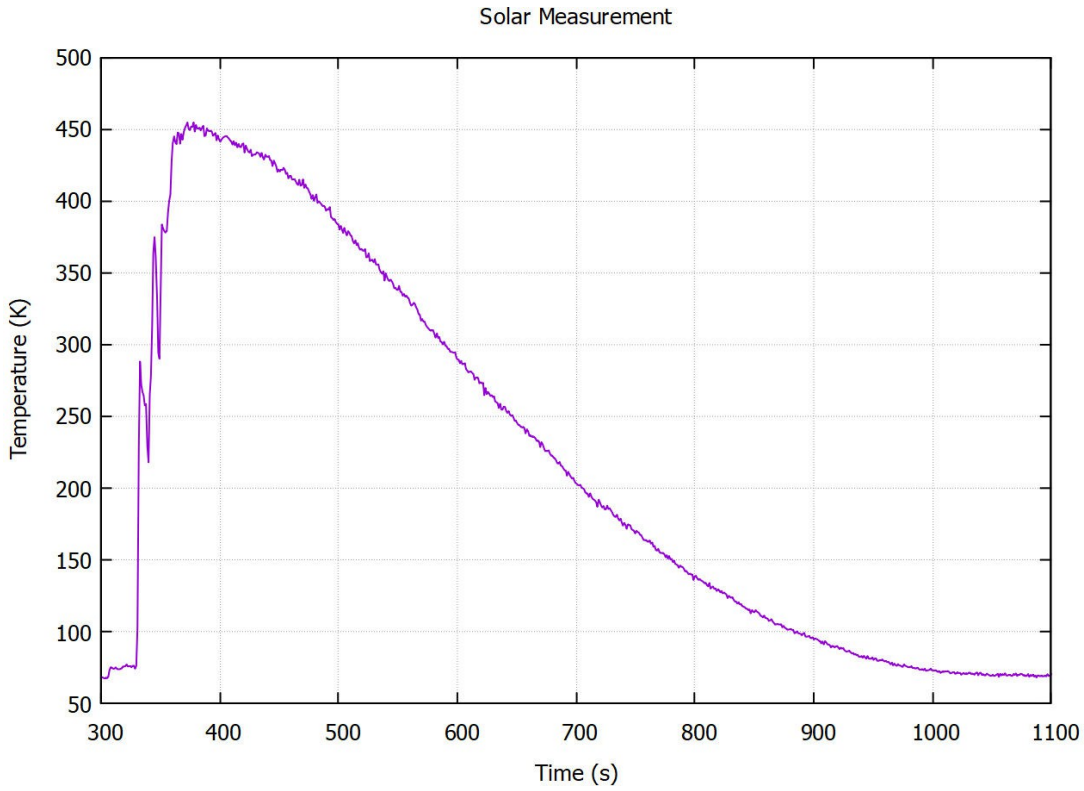

Figure 6.6: Solar temperature measurement on 31-10-2024 15:00 approx.

Next we`and pointed at the sun, in which we`and understood with the increase of the detected tempe- rature from 70K in the sky to 450K, as we expected from the experimental tests carried out last year, as shown in the cited article [13]. The bell-shaped trend shown`is due to the fact that, although we`and fixed the detector at a specific point, the earth turning sends the sun (450K detected[13]) out of the detection cone, whose antenna aperture and its directional gain not`and constant, but Gaussian type, as shown in the graph, gradually leading to detect only the sky (70K or so). The why we detect only 450K maximum by pointing at the sun`is due to the fact that since the dish has a conical aperture in which it detects more radio frequencies (called the main lobe of the directive antenna), the temperature we detect inside the cone`is actually`a the average of the temperature of the sun (11000K) and the cosmic background radiation (3K) that is detected in empty space, with added sky temperature of about 60-70K, as formulated in the article [13].

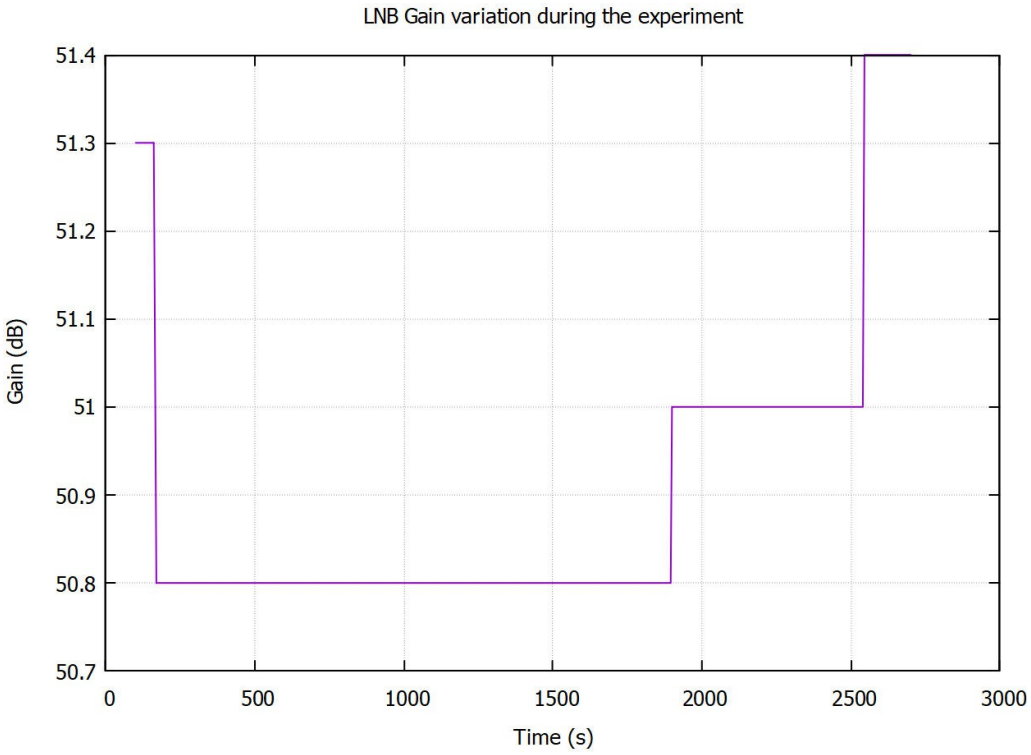

Figure 6.7: Change in gain going forward with the calibrations of the system at runtime, where one can clearly observe the decrease in LNB gain due to the LNB's increase in temperature during the sun's measurement, and its increase when it`and cooled, shortly after sunset.

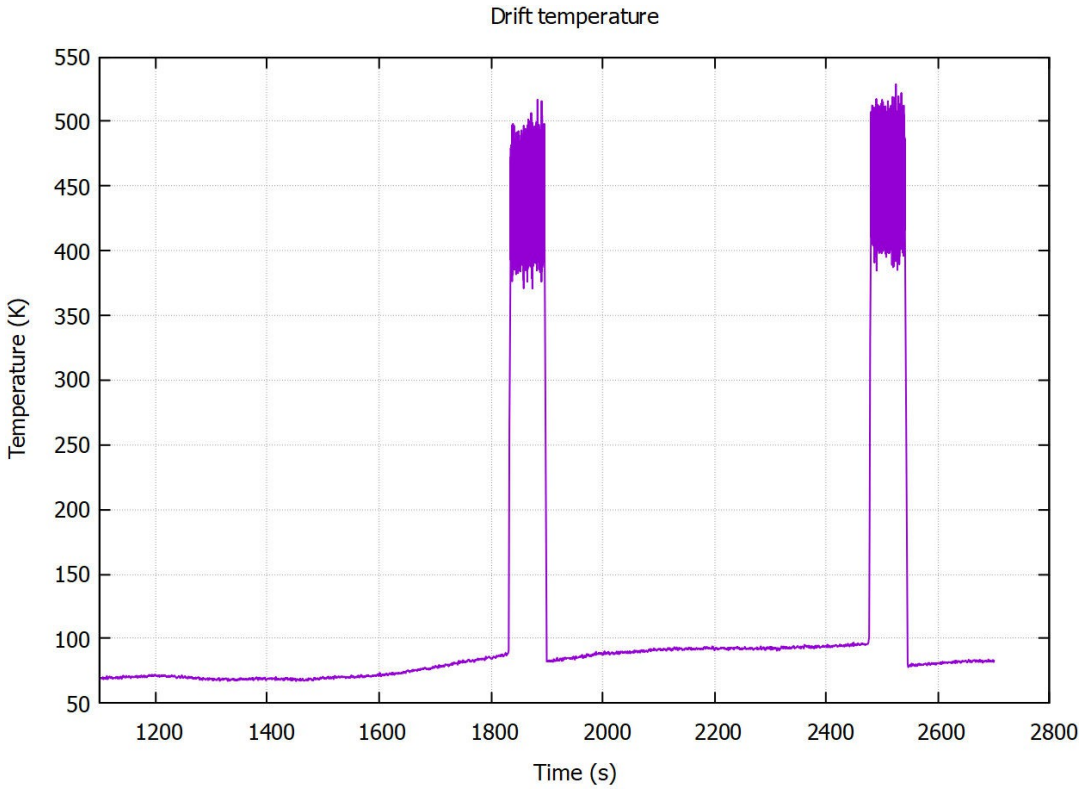

Figure 6.8: Temperature drift during thermal cooling of the LNB Interesting to observe

in the latter part of the measurement session that the temperature detected gradually increased, despite recalibrating more times. This phenomenon and due to the fact that the LNB, which was under the sun during measurement, reduced its gain by thermal drift. After the sun and set, therefore, the LNB began to cool down, returning to its original gain, showing a fictitious increase in the detected temperature due to this phenomenon.

Chapter 7

Conclusions

7.1 Conclusions on the results obtained

As we`and observed in the previous chapters, we`and could demonstrate the feasibility` a of using a simple and inexpensive piece of equipment (with a total cost of about 80€) to build and use a radiometer, making measurements that are quite accurate and in line with the expected theoretical values. In addition, ci see how, unlike the instrument designed
in the previous year, is also much nè compact, allowing it to be used
for purposes where the space`and very small, as well as being easily serializable and replicable due to its low total cost. In addition to this demonstration, it`e was also possible to characterize the SDR in various respects and accurately, succeeding in defining its dynamic rangeaccuracy, measurement fluctuations also due to temperature, frequency accuracy, and also its gain and noise figure, making it to all and purposes a usable instrument for making measurements, remaining within the limits of uncertainties due to the quality`a of the materials with which`it is composed and its intrinsic characteristics. The p , for`o, was`not only the p of making signal measurements, but rather noise measurements, since thanks to the use of an LNA (inside LNB), it`e was possible to minimize the nè dangerous contribution in the noise measurement, i.e.`e the noise figure. Allowing, as a result, the noise equivalent temperatures of the bodies pointed with the radiometer to be estimated particularly accurately, using the nè suitable parameters depending on the specific application.

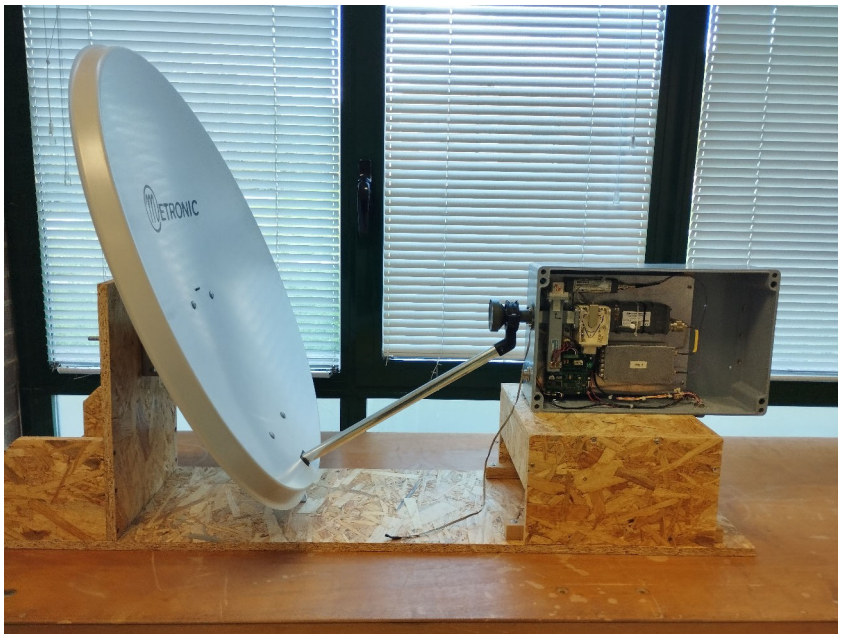

Figure 7.1: Radiometer built in 2023 [13]

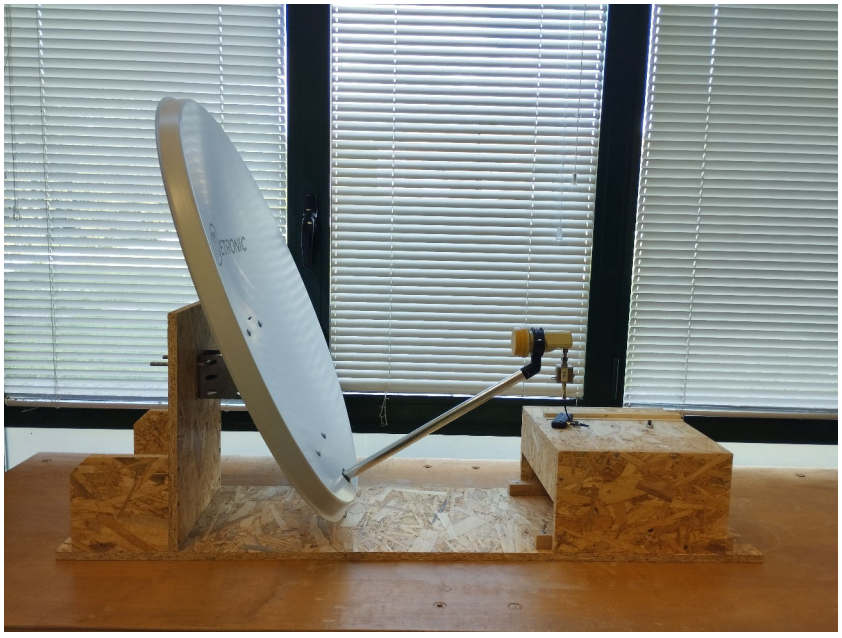

Figure 7.2: Current radiometer, consisting of dish, LNB, Bias-T and SDR

As you see, the difference in size is substantial, while still maintaining good accuracy and very low cost.

7.2 Future developments

The following describes future developments that are planned in order to improve the initial prototype, or give it applications in fields of concrete use.

7.2.1 Automation

One of the next steps will be to automate the calibration system through the black body, allowing the system to self-calibrate automatically by placing the black body itself in front of the illuminator, as is done by hand during the experiments described within this thesis, but allowing it to handle tighter timelines, allowing the possibility of calibrating the gain every N seconds for M seconds, with minimal delays. In order to be able to succeed in this task we have thought of automating the movement of the black body through the use of a servomotor controlled by a microcontroller, such as an Arduino, which will communicate with the rest of the system in such a way that it will operate only during the planned calibration phases, avoiding interference during the other phases.

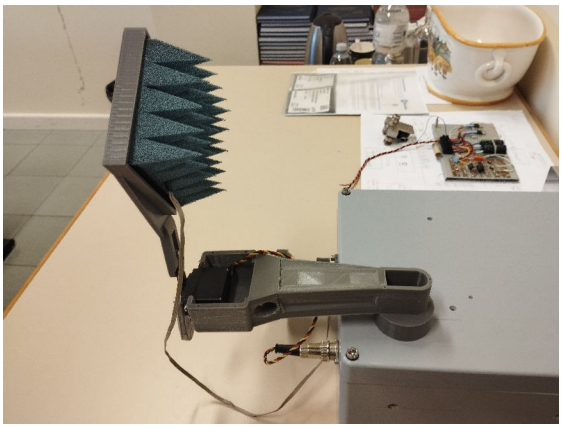

Figure 7.3: Automatic instrument calibration system with analog backend[13]

Figure 7.3 shows the system implemented in the old instrument with analog back- kend, which was, in fact, made up of a servomotor managed by a microcon- troller, which also managed internal components of the system, such as the temperature sensor to give *the* T_{BB} value to the main control system, including a radio receiver, to make the appropriate calculations in a fully automatic manner. This system operated with a timing of 3 seconds of measurement alternating with 3 seconds of calibration, to make it as stable as possible to temperature drifts

7.2.2 Use within CubeSat

One of the fields in which it was thought to be used`is within a Cubesat, a type of cubic satellite with a standard size of $1dm^3$ and a maximum mass of 1.33kg, or multiples thereof depending on how many modules, which each`and a cube of $1dm^3$, are installed, created in order to enable ï to be able to launch satellites into orbit, albeit temporarily, to carry out their studies and research in a particularly economical manner. In the case, the advantage would be in its compactness, since the volume occupied must be as small as possible, and also its low power consumption, which would make it ideal for carrying out radiomisurances of celestial bodies without having the obstacle of the atmosphere.

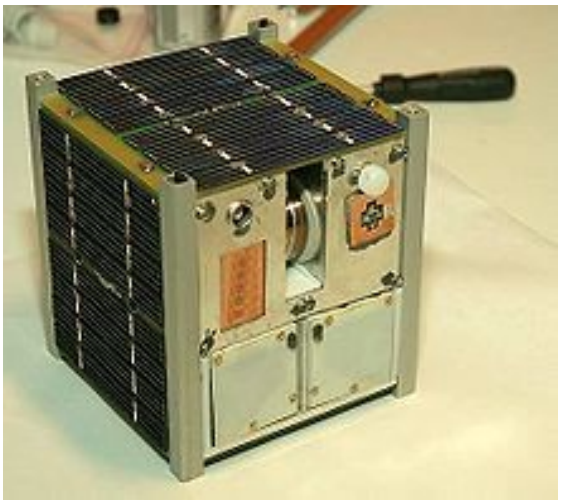

Figure 7.4: Example of CubeSat[2]

7.2.3 Further radio astronomy applications

Other possible radio astronomy applications would be the possibility`of replicating the instrument, being particularly inexpensive, in such numbers that the sun could be constantly observed by placing them at various points on the earth. Doing so could resolve the problem of the small observing window of a radiometer, which, if fixed à observe it for a maximum of 15 to 20 minutes and, if mobile, for as many hours as there are within the day depending on the season, allowing one to be always on the alert for major events, such as flares, in which à send the trigger to much nù accurate radiometers to accurately record the event. Currently our ìy is collaborating with the Technical Uìof Berlin (TUB) for this purpose.

Appendix A

Appendix

A.1 Python block script code

```
"""
Embedded Python Blocks:

Each time this file is saved, GRC will instantiate the first class it
    →finds
to get ports and parameters of your block. The arguments to__ init_____
    →will
be the parameters. All of them are required to have default values! """


import numpy as np
from gnuradio import gr
import pmt
import time
from pathlib import Path
import math


class blk(gr.sync_block): # other base classes are basic_block,
    →decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def__ init__ (self, frequency=1000000000, sampleRate=2048000,
        →timePeriod= 1, gain=30, gainLNB=52, offset=3, fileName=str(
        →time.time())+".txt", Trx=120, calibrationPeriod=60):
        """arguments to this function show up as parameters in GRC"""
```

```
gr.sync_block. init ( self,
    name='Integrator and file writer', # will show up in
        →GRC
    in_sig=[np.float32],
    out_sig=[np.float32, np.float32, np.float32]
)
# if an attribute with the same name as a parameter is found, #
a callback is registered (properties work, too).


self.startCalibrationPort = 'startCalibration'
self.message_port_register_in(pmt.intern(self.
    →startCalibrationPort))
self.set_msg_handler(pmt.intern(self.startCalibrationPort), self.
    →handle_msgStartCalibration)
self.startCalibration = False

self.endCalibrationPort= 'endCalibration'
self.message_port_register_in(pmt.intern(self.endCalibrationPort)
    →)
self.set_msg_handler(pmt.intern(self.endCalibrationPort), self.
    →handle_msgEndCalibration)
self.endCalibration = False

self.TbbPort= 'Tbb'
self.message_port_register_in(pmt.intern(self.TbbPort))
self.set_msg_handler(pmt.intern(self.TbbPort), self.handle_msgTbb
    →)
self.Tbb= 0

#Initialization parameters
self.freq= frequency
self.period= timePeriod
self.bandwidth= sampleRate
self.gain= gain
self.gainLNB= gainLNB
self.offset= offset
if fileName=="" :
    fileName=str(time.time())+".txt"
```

```
        self.Trx= Trx
        self.calibrationPeriod   =    calibrationPeriod

        self.path   =    Path(fileName)

        #Initializing support variables self.power= 0
        self.time 0   =
        self.calibrationTime 0    =
        self.fixedPower 0    =
        self.fixedTemperature 0    =
        self.temperature 0    =
        self.i= 0
        self.Pbb 0   =
        self.fixedCalibrationTime 0    =
        self.AvgPbb= 0
        self.tempTbb 0   =

        self.flagMeasurement= False
        self.startCalibration= False
        self.endCalibration= False
        self.flagFileHeader= True
        self.flagTbbSet= True

        #Defining constants self.absTime=
        time.time() self.kb= 1.380649 *
        pow(10, -23)



    def handle_msgStartCalibration(self, msg):
        if self.calibrationTime != 0 and not self.startCalibration:
            self.endCalibration= False
            self.calibrationTime= 0
        self.startCalibration= msg

    def handle_msgEndCalibration(self, msg):
        if not self.startCalibration and self.calibrationTime!=0:
            self.endCalibration= msg
```

```
def handle_msgTbb(self, msg):
    self.tempTbb=pmt.to_float(pmt.cdr(msg))

def work(self, input_items, output_items):

    if not self.endCalibration:

        if self.startCalibration:
            #When the calibration starts, I start the 60 count.
                →seconds
            if self.calibrationTime== 0:
                self.Pbb= 0
                self.i= 0
                self.calibrationTime= time.time()
                if self.flagFileHeader:
                    #Recreate the file in which it saves the data, at each
                        →session overwrites it with
                    self.path.open("w") as f:
                        f.write("")
                    #Write the
                    header
                    ="#Frequency: "+ str(self.freq)+ " Hz\n" string+=
                    "#Gain: "+ str(self.gain)+ " dB\n" string+=
                    "#Absolute time: "+ str(time.asctime(
                        →time.gmtime(self.absTime)))+ " UTC+2\n"
                    string+= "#Integration period: "+ str(self.period
                        →)+ " s\n"
                    string+= "#Calibration period: "+ str(self.
                        →calibrationPeriod)+ " s\n"
                    string+= "#LNB Equivalent Temperature: "+ str(
                        →self.Trx)+ " K\n"
                    string+= "#Offset: "+ str(self.offset)+ " dB\n" +=
                        '→    "##########################################\
                        →n"
                    string+= " time power temperature LNBGain\n"
                    string+= "#s dBm K dB\n"
                    string   +=
                        '→    "##########################################\
                        →n"
                    with self.path.open("a") as f:
```

```
                    f.write(string)
                self.flagFileHeader= False

            #I add samples as I go and mark the time passed if
            (time.time()-self.calibrationTime)<self.
                →calibrationPeriod:
                self.tempPbb =(pow(10,((input_items[0][0]-30)/10)))
                self.Pbb+= self.tempPbb
                self.fixedCalibrationTime = time.time()-self.
                    →calibrationTime
                self.temperature = self.tempPbb/(self.kb*self.
                    →bandwidth*(pow(10,((self.gainLNB)/10))))
                self.fixedTemperature= self.temperature
                self.fixedPower= 10*math.log(self.tempPbb,10)+30
                string=" " + '{:.2f}'.format(round(time.time()-
                self.
                    →absTime, 2))+ " "+ '{:.2f}'.format(round(10*
                    →math.log(self.tempPbb,10)+30, 2))+ " "+ '{:.1f
                    '→ }'.format(round(self.temperature, 1)) + " " +
                    → '{:.1f}'.format(round(self.gainLNB, 1))+ "\n" with
                self.path.open("a") as f:
                    f.write(string)

                self.i+= 1

            #When 60 seconds have passed, I average the
                →power
            if (time.time()-self.calibrationTime)>=self.
                →calibrationPeriod:
                self.startCalibration= False
                self.AvgPbb= self.Pbb/self.i
                self.flagTbbSet= True

    else:
        if self.flagTbbSet:
            self.Tbb=self.tempTbb
            self.gainLNB= 10*math.log((self.AvgPbb / (self.kb*(self.
                →Tbb+self.Trx)*self.bandwidth)), 10)
            self.flagTbbSet= False
        #Save the start measurement time of the specific band if not
        self.flagMeasurement:
```

```
            self.time= time.time()
            self.flagMeasurement= True
            self.i= 0

        #Integro the new champion
        if self.flagMeasurement and (time.time()-self.time)< self.
            →period:
            self.power+= pow(10,((input_items[0][0]-30)/10))
            self.i+= 1

        if self.flagMeasurement and (time.time()-self.time)>= self.
            →period:
            self.power= self.power / self.i
            Y= self.power/self.AvgPbb #Y factor self.temperature    =
            ((self.Tbb+self.Trx)*(Y))-self.Trx
            #self.temperature   =    (pow(10,((self.power-30)/10)))/(self.
                →kb*self.bandwidth)
            #Prepare string and write to file.
            string=" "     +   '{:.2f}'.format(round(self.time-self.absTime,
                '→   2)) " "  +      +   '{:.2f}'.format(round(10*math.log(self
                →.power, 10)+30, 2))+ " "+ '{:.3f}'.format(self.
                →temperature)++'{:.1f}'.format(round(self.
                →gainLNB, 1))+ "\n"
            with self.path.open("a") as f:
                    f.write(string)

            #Average power over integration time.
                →set in the current band self.fixedPower=
            10*math.log(self.power,10)+30 self.power= 0
            self.fixedTemperature= self.temperature
            self.temperature= 0

            #Save the time taken for measurement in the self.time band= 0
            self.flagMeasurement= False


        #Mostro the power of the last measurement made, the
            →frequency, time current iteration
```

```
output_items[0][:]= self.fixedPower
output_items[1][:]= self.time-self.absTime
output_items[2][:]= self.fixedTemperature
output_items[3][:]= self.fixedCalibrationTime return
len(output_items[0])
```

A.2 Demonstration calibration through T_{hot} and T_{cold}

Assuming:

$$\begin{cases} T_{hot} = T_{BB} \\ T_{cold} = T_{sky} \end{cases} \tag{A.1}$$

That`and we use the equivalent black body *temperature* T_{BB} as the warm temperature and the equivalent sky *temperature* T_{sky} as the cold temperature (other options are also valid,such as liquid nitrogen, in case they are available), which are known variables by exploiting a thermometer in the black body case. The system of equations we derive from the thermal noise power equations of a receiver system`and the following:

$$\begin{cases} P_{BB} = k_B (T_{BB} + T_{LNB}) GLNBB \\ P_{sky} = k_B (T_{sky} + T_{LNB}) GLNBB \end{cases} \tag{A.2}$$

Where B`and the bandwidth of the SDR receiver, $k_B$ the Boltzmann constant, and $P_{BB}$ and $P_{sky}$ we obtain by making the above measurements of the targets for a time that can`or vary from a few seconds to several minutes, depending on the need and the entity`a of the system drifts, and then averaging them arithmetically, to reduce noise deviations. By expressing T_{LNB} in the first equation we obtain:

$$\begin{cases} T_{LNB} = \dfrac{P_{BB}}{k_B G_{LNB}} - T_{BB} \\ P_{sky} = k_B (T_{sky} + T_{LNB}) GLNBB \end{cases} \tag{A.3}$$

Plugging the equation of T_{LNB} into the second equation gives:

$$\begin{cases} T_{LNB} = \dfrac{P_{(BB,}}{k_B G_{LNB}} - T_{BB} \\ GLNB = \dfrac{P_{sky} - P_{BB}}{k_B B T_{sky}} \end{cases} \tag{A.4}$$

Finally, by substituting the second equation of A.4 into the first:

$$\begin{cases} T_{LNB} = T_{sky} \left(\dfrac{P_{BB}}{P_{BB} - P_{sky}} \right) - T_{BB} \\ GLN = \dfrac{P_{sky} - P_{BB}}{k_B T_{sky} B} \end{cases} \tag{A.5}$$

With A.5 we demonstrate the proof of being able to calibrate particularly accurate the instrument, simply by making indirect measurements of temperature and target power, the proof which makes even an instrument in itself` and inaccurate because of the quality`a of the components, a good candidate for this type of measurement.

Bibliography

[1] George Goussetis Alexios Costouri, James Nessel. *Validation of a Digital Noise Power Integration Technique for Radiometric Clear Sky Attenuation Estimation at Q-Band*, 2020.

[2] Wikipedia contributors. Cubesat - wikipedia. https://it.wikipedia.org/wiki/CubeSat, 2024. Online; accessed 12-November-2024.

[3] Wikipedia contributors. Low-noise block downconverter - wikipedia. https://en.wikipedia.org/wiki/Low-noise_block_downconverter, 2024. Online; accessed 7-November-2024.

[4] Wikipedia contributors. Python - wikipedia. https://it.wikipedia.org/wiki/Python, 2024. Online; accessed 8-November-2024.

[5] Wikipedia contributors. Radiation absorbing material - wikipedia. https://en.wikipedia.org/wiki/Radiation-absorbent_material, 2024. Online; accessed 9-November-2024.

[6] ESA. Esa - cimr. https://www.esa.int/ESA_Multimedia/Images/2020/11/CIMR, https://www.esa.int/Applications/Observing_the_Earth/Copernicus/CIMR, 2024. Online; accessed 30-October-2024.

[7] Md Mehedi Farhad, Ahmed Manavi Alam, Sabyasachi Biswas, Mohammad Ab- dus Shahid Rafi, Ali C. Gurbuz, and Mehmet Kurum. *SDR-Based Dual Polarized L-Band Microwave Radiometer Operating From Small UAS Platforms*, 2024.

[8] GNUPlot. Gnuplot. http://www.gnuplot.info/, 2024. Online; accessed 8-November-2024.

[9] IZ0ABD. Bias tee for remote amplifier. https://iz0abd.wordpress.com/2021/02/10/bias-tee-for-remote-amplifier/, 2024. Accessed: 7-November-2024.

[10] Joan Francesc Munoz-Martin, Lara Fernandez Capon, and Joan Adria Ruiz de- Azua an Adriano Camps. *The Flexible Microwave Payload-2: A SDR-Based GNSS-Reflectometer and L-Band Radiometer for CubeSats*, 2020.

[11] GNU Radio. Gnu radio. https://www.gnuradio.org/, 2024. Online; accessed 8-November-2024.

[12] RTL-SDR.com. About rtl-sdr. https://www.rtl-sdr.com/about-rtl-sdr/, 2024. Online; accessed 6-November-2024.

[13] Giacomo Schiavolini, Giulio Brancali, Ethan Bernardini, Valentina Palazzi Giu- lia Orecchini, Camille C.A. Westerhof, Timo S. Prinz, Sebastian Lange Mar- tin Hübner, Maurizio Burla, , and Federico Alimenti. *Low-Cost Calibrated Mi- crowave Radiometers for Solar Observation: from Education to Science*. Univ of Perugia and Technische Univ Berlin, 2024.

[14] GNU radio wiki contributors. Gnu radio wiki. https://wiki.gnuradio.org/index.php/Main_Page, 2024. Online; accessed 7-November-2024.

[15] Wikipedia. Microwave radiometer - wikipedia. https://en.wikipedia.org/wiki/Microwave_radiometer, 2024. Online; accessed 6-November-2024.

Appendix B

Acknowledgements

First of all, I would like to thank my thesis advisor, Prof. Federico Alimenti, who stood by me throughout the thesis, always making himself available for help, explanations on things that were less clear to me and getting me more and more interested in the field of radio frequencies, and Giacomo Schiavolini, who gave me important tips on the practical and final part of the thesis, when the tests and experiment were carried out. In addition, I also want to thank my parents, my brother and the rest of the family for always believing in me and never pressuring me in any way during these 3 years, always supporting me. As I also thank all my group of friends, both from Perugia and Frosinone, that I made both in and out of the university, for all the study sessions, as well as the evenings, spent together, and it has always been nice and I am happy to have met all of you, these years have been among the nicest ones I have ever had and I hope we will stay in touch afterwards, when everyone will take their own way. Finally, I would also like to thank the professors at ITTS A. Volta in Perugia, who formed me very well and made me understand what path I wanted to take. Thanks to everyone and thank you to everyone who has been close to me during these years, without you I don't think I would be the person I am now.