POLITECNICO DI TORINO

Collegio di Ingegneria Elettronica, delle Telecomunicazioni e Fisica

Master of Communications and Computer Networks Engineering

Master Degree Thesis

Innovative microwave rain gauge operating at 77 GHz: software development and operational test

Supervisor:
Ing. Marco Allegretti

Co-Supervisors:
Ing. Silvano Bertoldo
Prof. Giovanni Perona

Candidate:
Luca Bongarzone

Contents

List of tables

List of figures

Chapter 1

Introduction

1.1 Objectives

The thesis aims to show that, with specific assumptions and theoretical background, it is possible to use a 77 GHz (W-band) radar as a mini weather radar and/or as a microwave rain gauge, with high-resolution, to be used for various purposes including Quantitative Precipitation Estimation (QPE), and usage on a car, considering the concept of car-as-sensor.
Objectives of the thesis are the following:

1) Derive the relationship between rainfall rate (R) and radar reflectivity factor (Z) considering the scattering theory at 77 GHz.
2) Realization of a fully functional 77 GHz radar prototype for rain monitoring, taking as basis one of the standard W-band radar present on the market.
3) Development of a software to control and manage the prototype.
4) Test of the prototype.

1.2 Car as sensor

According to European Telecommunication Standards Institute (ETSI), the W-Band is not designated for metereological purposes, but it has actually been standardized for automotive applications and, in particular for automatic Cruise Control radar, operating at around 77 GHz. The idea is indeed, to exploit, as a future development, a common 77 GHz radar, mounted on cars, for rain monitoring, in order to use a car as a moving integrated weather sensor.
In such a way, there is a clear "car as sensor" concept extension, in a wider view of Internet Of Things (IOT): nowdays, cars are infact equipped with a large set of sensors, systems and technologies, commonly used to improve car and passengers'safety and comfort. These sensors have two main advantages, in data acquisition: since cars have not any constraints with their power source, they can be easily equipped with powerful processing capabilites; also, since cars have not any constraints on space and capability, they can be equipped with sufficient storage units.

Some examples of sensors already installed on cars, to inquire information about the environment, and that can so be potentially used for metereological purposes include: accelerometers, external temperature sensors, pressure sensors, cameras and so on.
A set of cars equipped with weather radar can constitute a capillary network for rain estimation, providing information in real-time about the amount of precipitation in a given area.
Infact, exploiting the new interfaces and standars (802.11p) for Veichle-to-Veichle (V2V) and Veichle-to-Infrastructure (V2I) communication, it would also be possible to exchange and process a large amount of data in relatively short period of time.

Figure 1. "Car as Sensor"

Alternatively to the car's application, the 77 GHz radar could be used for rain monitoring in particularly difficult areas, where there eventually are just long range and small resolution radar whose coverage is not too much precise. For example in mountanious environments there are always areas without radar coverage due to natural obstacles and this could be solved thanks to these 77 GHz radar and their high resolution.

Chapter 2

Radar Systems

Radar is the definition for a generic electronic system able to simultaneously transmit electromagnetic waves (EM) in radio frequency (RF: 3 Hz – 300 GHz more or less) through an interest region, and to detect the ones reflected from surfaces within the region itself. Surfaces that can typically be differentiated in: targets, if related to something that needs to be individuated by the radar (like airplanes, boats, atmospheric phenomena, etc.), and clutters, if related, instead, to obstacles or something undesired causing interference.

Even if the characteristics of a radar system vary according to the use application, its main components include:

- Transmitter, to generate high power EM waves able to induce electric currents, on the surfaces situated along the waves' propagation path, that produce the scattering radiation, individuated by the radar.
- Antenna, to physically introduce the generated waves in the propagation medium, separating transmitted and received signals.
- Receiver, to process EM waves individuated by the antenna: usually the signal is amplified, converted to an IF suitable for the A/D conversion, and demodulated.
- Signal elaborator, to distinguish interference (electronic and thermal noise, internal and external interference, like clutters and jamming) from the signal of interest.

The following **Figure 2** shows a typcal (and generic) example of radar diagram block:

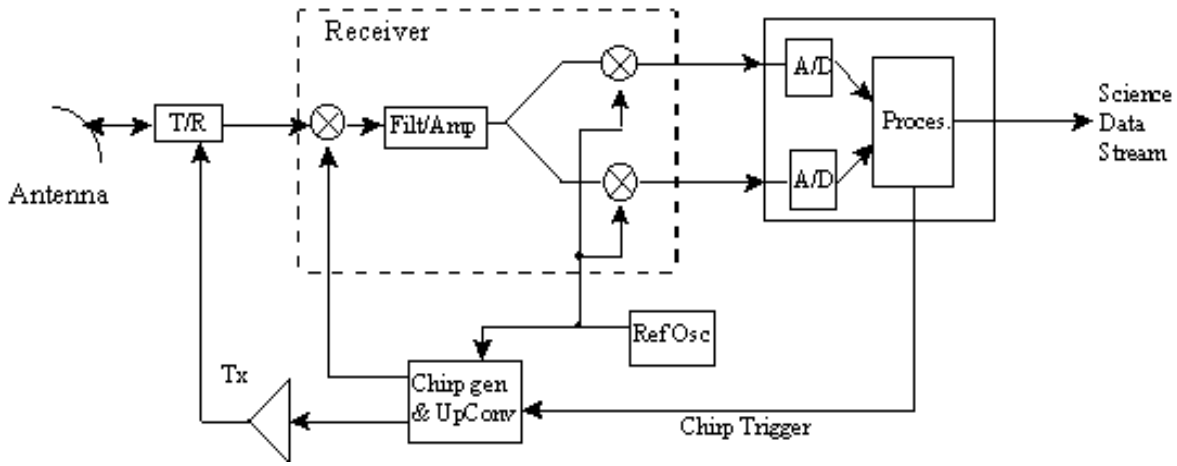


Figure 2. Radar diagram block

There are two possible basic configurations for the radar systems antennas: monostatic, in which a single antenna is adopted for both transmitter and receiver (more common nowdays), and bistatic, in which two different antennas are attached to the two devices.
Another classification for the radar systems concerns the employed waveform transmitted, that can be impulsive or continuous (CW). In the first case, either the transmitter emits a finite pulse sequence and the receiver is isolated from the antenna, or the transmitter is turned off and the receiver is ready to detect the reflected signals. In the second case, transmitter and receiver are continuously operative.

2.1 FMCW Operating principles

Since the 77 GHz radar I used belongs to the CW radar category, and in particular to the FMCW (Frequency Modulated Continuous Wave) one, I'm going to focus on them. This choice depends on the fact that these types of radar are already mounted on the car and consume much less power than the pulsed radars.
The main FMCW radar principle is that the frequency of the transmitted EM signal changes over time, generally with a sweep across a set bandwidth "BW". A sawtooth function is the simplest, and most often used, change in frequency pattern for the emitted signal. Synthesized modules (like RS3400W I used for this thesis), actually, don't sweep the frequency continuously, but rather step it with a set of discrete points, thus they are also called SFCW (Stepped Frequency Continuous Wave) radar. The signal sources, ensure very precise frequency control, fundamental for the accuracy and the repeatability of the measurements; the transmitted power can be very low with respect to other types of radar.
Given a frequency sweep time "T", during which the sweep must be completed, it's possible to define the sweep rate "k_f":

$$k_f = \frac{BW}{T} \tag{1}$$

Since a delay caused by the time of flight of the reflected signal is introduced, depending on the position "d" of the scatterer and on the speed of light "c", the total time difference is Δt :

$$\Delta t = \frac{2d}{c} \tag{2}$$

The difference in frequency between the transmitted and the received signal, at a given time instant, is determined by mixing the two signals and producing a new one, whose frequency, called "beat frequency" putting together (1) and (2) is:

$$\Delta f = k_f \Delta t = \frac{BW}{T}\frac{2d}{c} \tag{3}$$

According to Werner formula, indeed, the product between two sinusoids (the RF signal transmitted and the received one) is equal to ½ cosine of the sum of their frequencies (eliminated by low pass filter), times the cosine of their frequencies difference. This remaining cosine is the IF signal that the radar outputs, constant and with the frequency Δf depending on the target echo distance. This procedure is repeated for every individuated echo, thus the final IF signal outputted from the radar will contain superpositions of the individual IF signals and so different frequency components (one per each echo) that can be used to measure distance or velocity of the target.

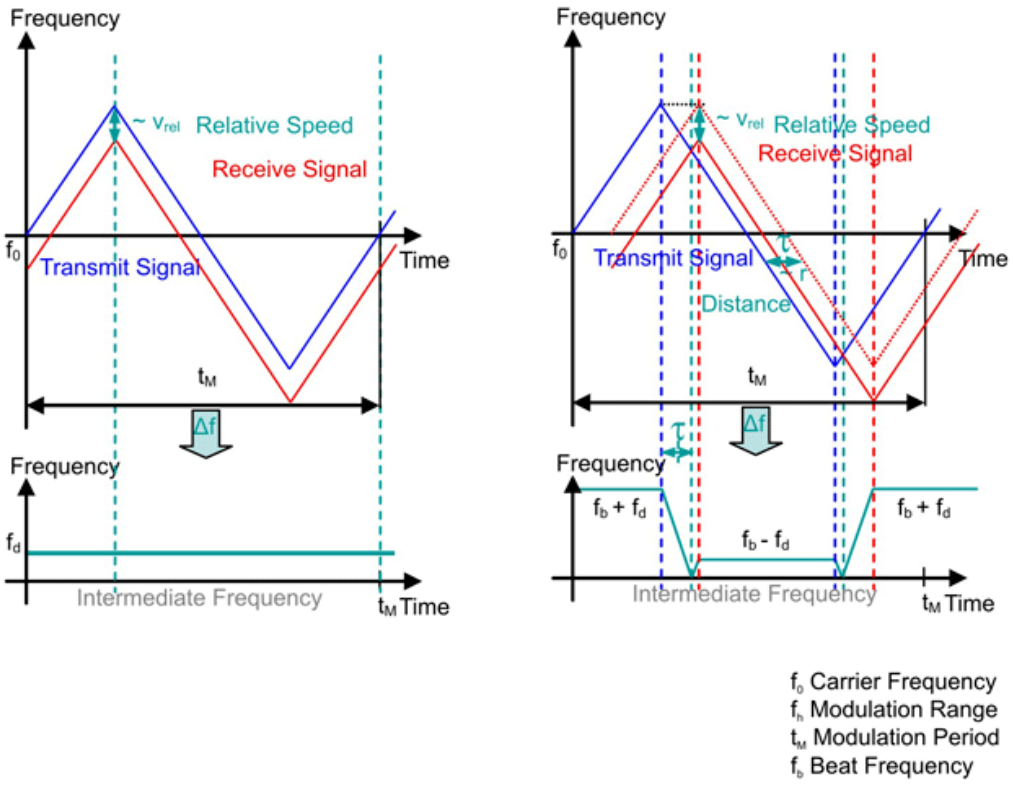


Figure 3. FMCW Signal

2.2 FMCW Characteristics

Since the signal coming from the radar is a sinusoid with multiple frequencies, after computing its Fourier Transform, it's possible to individuate a target distance "R_f" by simply looking at the power peaks and their corresponding frequencies "f":

$$R_f = \frac{cTf}{2BW} \tag{4}$$

According to the mentioned parameters, it is possible to calculate the radar range resolution:

$$Res = \frac{c}{2BW} \tag{5}$$

This shows how it only depends on the sweep bandwidth of the RF signal and not on its operating frequencies. Thus by increasing it, it's possible to reach very high resolutions (as it happens for the RS3400W, since it works at 77 GHz and a high bandwidth can be exploited). Targets at lower distances than the resolution can't be considerated as valid, since at least one complete frequency sweep must be performed.
The theoretical maximum distance that can be reached by the radar, without ambiguity (since otherwise two targets over that range can contribute to the spectrum at the same frequency and there can be no unique determination of range) is:

$$R_{unambiguous} = \frac{cT}{2} \tag{6}$$

Obviously, the resulting value should be much lower since different attenuation factors come up.

2.3 FMCW Radar equation

Target detection for a FMCW radar (and more in general, all the radar systems) is based on its echoes' intensity measurement, which is directly proportional to the radar cross section (RCS), also known as Backscattering Cross Section, of the target itself.

Generally, the RCS is the ratio between the power reflected from a scatterer and the power incident on it, and measures how much it is detectable by a radar: the larger the RCS of a target, the higher are the probabilities to detect it.

For a monostatic radar and a puntiform scatterer, the received power P_r is calculated as function of the transmitted power P_t, the transmitter and receiver antenna gains G_t and G_r, the transmitter wavelength used λ, the target distance R and the scatterer cross section σ_o:

$$P_r = \frac{P_t G_r G_t \lambda^2 \sigma_o}{(4\pi)^3 R^4} \tag{7}$$

Considering a volumentric target instead, it turns out that the total received power from the radar depends on the total Backscattering section σ_w, associated with scatterers (since usually in a radar volumetric cell, more scatterers are present), contained in the volume V, monitored from the radar, that is equal to the product between V itself and the radar reflectivity η:

$$\sigma_w = \eta V \tag{8}$$

η $[\frac{m^2}{m^3}]$ represents the RCS per volume unit (1 m^3) dV, and it is given by the sum of each RCS σ_i detected, within dV:

$$\eta = \frac{\sum_i \sigma_i}{dV} \tag{9}$$

The total volume V, illuminated by a CW radar, can be instead expressed as follows:

$$V = \frac{\pi * \theta_{3dB} * \varphi_{3dB} * R^2}{8 * ln2} * \frac{c}{2BW} \tag{10}$$

being θ_{3dB} and φ_{3dB} the Half Power Beam Width (HPBW) angles of the radar antenna, in the two reference orthogonal radiation planes.

Substituting, in the equation (7), σ_o with σ_w and its expression (8), combining (9) and (10), the expression for the total received power is:

$$P_r = \frac{P_t\ c\ G_r G_t \lambda^2 * \theta_{3dB} * \varphi_{3dB} * \eta}{1024 * BW * \pi^2 * ln2 * R^2} \quad (11)$$

2.3 FMCW Problematics

As for any other radar system, the FMCW ones are subjected to different problematics, linked to the EM waves propagation phenomena: reflection, refraction, diffraction and attenuation.
The reflection is the basic of the radar theory and without it, it wouldn't be possible to individuate targets. However, there could be undesired reflections due to obstacles, and so clutters, that should be discovered and then filtered out in the signal post processing.
The refraction comes up at the interface between two materials, as for example, two different strates of the atmosphere, and consists in the path deviation of the EM waves.
The diffraction is still a path deviation of the EM wave, but it's caused by the presence of an obstacle that can be penetrated by the EM wave itself. This phenomenon is more evident when the obstacle has dimensions comparable or rather smaller than the incident wavelength.
The attenuation is the most important one and it's strictly linked to the absorption. The atmosphere is indeed made up of different gas and particles that could cause energy losses in the EM waves. Absorption is dependent on the signal frequency and on the target distance.
Since the purpose of this thesis is to show the possibility to adopt a 77 GHz radar as a meteorological sensor able to measure the amount of rain fallen in a short range, some of this propagation phenomena can be neglected, such as absorption and refraction. Ranges lower than 100 meters, indeed, exclude the possibility to cross different atmospheric strates, and the absorption, thus the attenuation, usually has impact whenever the signal reaches at least 1 km of distance.
Undesired reflections and diffraction phenomena should be instead kept in account, as well, as possible interferences. However, by post processing the signal, it can be easily individuated if some anomalies have come up during the measurements.

Chapter 3

Metereological Radar

3.1 Rain drops scattering

Rain estimation, through radar equipment, is based on the power received (11), considering as scatterers for the total RCS σ_w evaluation, the rain drops present in the volumetric cell, monitored by the radar itself, during its scan interval.

The rain drop RCS "σ_i", and the attenuation it introduces, is a function of its diameter D[mm] and of the wavelength (thus of the frequency) used by the radar. Defining the size parameter as:

$$\chi = \frac{D\pi}{\lambda} \tag{12}$$

It is possible to distinguish three different scattering regions:

- $\chi \ll 1$: Rayleigh's scattering, where the particles are smaller than the incident wavelength.
- $\chi \ll 1$: Mie's scattering, where the particles have dimensions comparable with the ones of the incident wavelength.
- $\chi \gg 1$: Geometric's scattering, where the particles are greater than the incident wavelength.

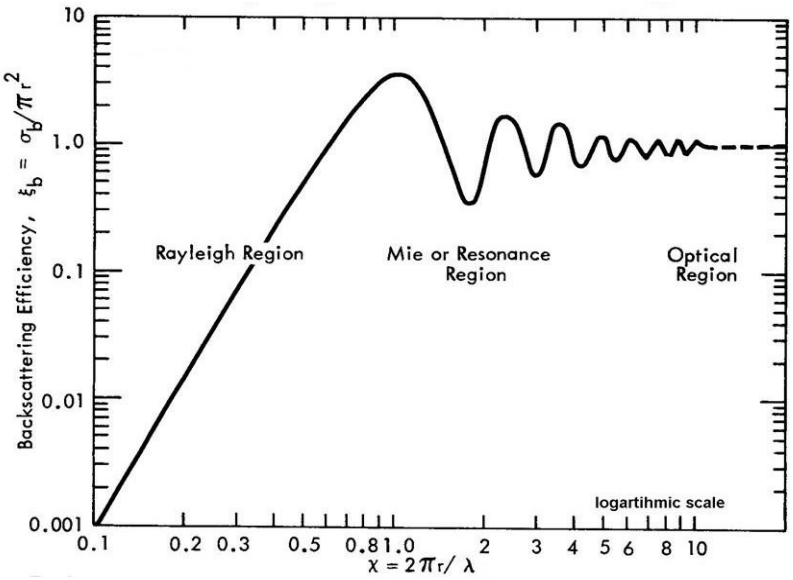


Figure 4 Scattering regions

Defining also the Backscattering efficiency "ξ_b" as the ratio between the RCS "σ" of a perfect sphere (solid that can approximate a rain drop) of radius r ($\sigma = \pi r^2$), and its surface (πr^2), it can be seen on the above **Figure 4** how it varies with the size parameter (in logarithmic scale). In radar metereology, the common approach is to assume the Rayleigh's model, which is valid just until the rain drops' physical dimensions are lower than the wavelength used. If they are comparable with it, instead, the Mie's model should be considered, since it gives a complete and mathematically rigorous solution of the scattering problem when an electromagnetic wave hits a sphere.

However, the standard bands used for meteorological purposes are the following:

- S-band (f=2.7-2-9 GHz, λ = 10-11 cm) used even for big hail particles (d > 2cm).
- C-band (f=5.6-5.65 GHz, λ = 5 cm) used even for large snow particles (d > 1cm).
- X-band (f=9.3-9.5 MHz, λ = 3 cm) used even for common hail and snow particles.
- Ka-band (f=35 GHz, λ < 1 cm) used for rain drops having diameter d > 0.1 mm.

Frequency	wave-length	band	meteorological application
20-300 MHz	1-15 m	**VHF**	wind profiler and ocean surface motion
400-900 MHz	0.3-0.7 m	**UHF**	wind profiler
1 GHz	0.3 m	**L-band**	boundary layer wind profile
2-4 GHz	7-15 cm	**S-band**	long-range precipitation radars
4-8 GHz	4-7 cm	**C-band**	long-range precipitation radars
8-16 GHz	2-4 cm	**X-band**	precipitation radars, marine radars
16-20 GHz	1-2 cm	**Ku-band**	radars
35 GHz	8.5 mm	**Ka-band**	precipitation and cloud radars
90-100 GHz	3 mm	**W-band**	cloud radars, Mie minimum

Figure 5. Common radar bands in meteorology

The above mentioned bands can correctly use the Rayleigh's model except for the Ka-Band where the wavelength (decreasing with the increasing of the frequency) begins to be comparable with the rain drops' diameter.

3.2 Rayleigh's model

Considering a certain rain rate R[mm/h], the drop size distribution (DSD) can be generally expressed as:

$$N(D) = N_0 D^{\mu}\, e^{-\gamma D} \tag{13}$$

$N(D)$ being the rain drop amount per unit of volume and dimensional interval $[mm^{-1}m^{-3}]$, D the drop diameter [mm], N_0 $[mm^{-1}m^{-3}]$ a parameter of the distirbution and $\gamma = \alpha R^{\beta}$ a parameter varying with the precipitation type.

Marshall and Palmer proposed one of the actual most common DSD used in metereology:

$$N(D) = 8000\, e^{-D\,(4{,}1 * R^{-0{,}21})}$$

Joss proposed instead a different one for thunderstorms:

$$N(D) = 1.4 * 10^{6} * e^{-D\,(3000 * R^{-0{,}21})}$$

The following **Figure 6** shows how, independently from the DSD, most of the drops have a diameter included between 0.5 mm and 4 mm.

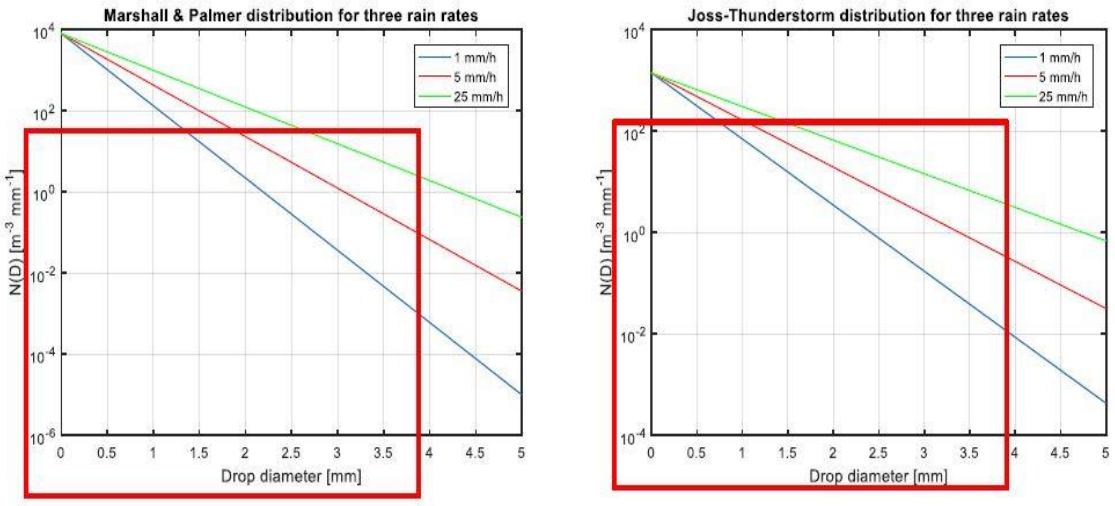


Figure 6. Drop size distribution

According to the Rayleigh's model, the Backscattering cross section associated with the i-th rain drop can be expressed as:

$$\sigma_i = \frac{\pi^5 |K^2| D_i^6}{\lambda^4} \tag{14}$$

Where $K = \frac{m^2-1}{m^2+1}$ being m the complex refraction index of the scatterer (water in this case).
Since in a rain cell there are multiple rain drops, the equation (9) must be evaluated substituting σ_i with its expression (14), such that:

$$\eta = \frac{\pi^5 |K^2|}{\lambda^4} \frac{\sum_i D_i^6}{dV} \tag{15}$$

Now it's possible to define the radar reflectivity factor $Z[mm^6 m^{-3}]$, representing here the drops concentration in the volume unit, as a function of drops diameter and distribution. Z is an intrinsic property of the rain drops, thus it depends on the precipitation type and dosen't depend on the wavelength used, as instead η does. So, given a certain DSD $N(D)$, according to the Rayleigh's model it is equal to:

$$Z_{Ral} = \int_0^{D_{Max}} D^6 N(D) dD \tag{16}$$

This expression substitute $\frac{\sum_i D_i^6}{dV}$ in the equation (15). So the final RCS per unit volume is:

$$\eta = \frac{\pi^5 |K^2|}{\lambda^4} Z \tag{17}$$

Therefore the radar equation (11) for metereological purposes, according to Rayleigh's model, substituing η with the expression of equation (17) and keeping also in account all the adjunctive attenuation factors L, turns to be:

$$P_r = \frac{P_t \, c \, G_r G_t \lambda^2 * \theta_{3dB} * \varphi_{3dB} * L^2}{1024 * BW * \pi^2 * ln2 * R^2} \frac{\pi^5 |K^2|}{\lambda^4} Z = \frac{Konst * Z}{R^2} \tag{18}$$

where R^2 is the squared target distance (in Km) and $Konst$ is the radar constant. In the following paragraph (3.4) I'll focus on $Konst$ characteristics.
Usually, the received power is expressed in dBm therefore a logarithmic scale conversion is needed, as shown below:

$$Pr|_{dBm} = 10log_{10}(P_t) + 10log_{10}(Konst) + 10log_{10}(Z) - 180 - 20log_{10}(R) \tag{19}$$

The factor 180 has to be subtracted, since the radar reflectivity factor is multiplied by 10^{18} to correctly express the measuring units (Power in [dBm], R in [km], Z in [mm^6/m^3]).

3.3 Mie's model

Since the radar I am using is working at 77 GHz (W-band) and the corresponding wavelength has a dimension of about 4 mm, comparable with the rain drops dimension, the Rayleigh region cannot be considered valid anymore. It is necessary to use the Mie's model, otherwise the backscattering efficiency, thus the total RCS, could be overestimated by up to 20 dB, as shown in the following **Figure 7**:

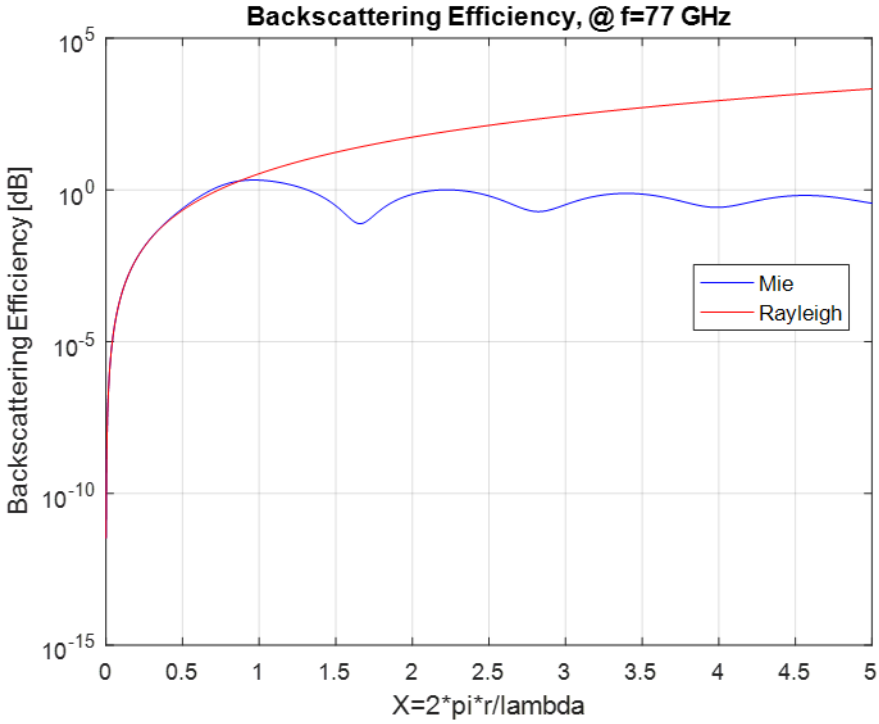


Figure 7. Backscattering efficiency

Using Mie's model implies to consider raindrops as perfect spheres and, although it is a rough approximation of the observations, it provides a first good estimate of rain drops shape that does not affect the results too much. However, even if it is more complicated to evaluate the backscattering RCS, whose value depends on Mie's coefficients, the radar reflectivity factor according to the Mie's model can be expressed as:

$$Z_{Mie} = \frac{\lambda^4}{\pi^5 |K^2|} \int_0^{D_{Max}} \sigma_{Mie} N(D) dD \tag{20}$$

It is worth noting that, according to this formulation, the radar reflectivity factor is a function of the wavelength, unlike the Rayleigh approximation, in which it is frequency independent.

3.4 Radar Constant

In this paragraph I am focusing on $Konst$ characteristics, the last factor of the equation (18). The radar constant is evaluated by the following expression:

$$Konst = \frac{c\, G_r G_t\, \pi^3\, \theta_{3dB}\, \varphi_{3dB}}{1024\, BW\, \lambda^2\, ln2} L^2 K^2 \tag{21}$$

where:

- $\frac{c}{2BW} = Res$ as shown in the second chapter with (5).
- G_r and G_t are the receiver and transmitter gains, as shown with equation (7).
- λ is the radar transmitted wavelength.
- θ_{3dB} and φ_{3dB} have already been illustrated with equation (10). They can be derived inverting the following relationship, valid for an elliptical horn (as the one I used, as explained in the following chapter), approximating the two angles as equal:

$$G_t\,|_{linear} = \frac{16}{sin(\theta_{3dB})\, sin(\varphi_{3dB})} = \frac{16}{sin^2(\theta_{3dB})} \tag{22}$$

- K has been mentioned above with equation (14) and depends on dielectric properties of rain (e.g. relative permittivity that takes into account the wavelength). For a 77 GHz it can be estimated equal to 0.75, while with classical Rayleigh's approach it has the value of 0.93.
- L includes the attenuations due to atmospheric absorption (they are like 1dB/km, since the W-Band is located in correspondence of a relative minimum of the atmospheric absorption curve, as can be noticed from the following **Figure 8**) and atmospheric attenuation due to rain (they are equal to 0.07 dB/km due to dry air and 0.36 dB/km due to water vapor component, considering the International Standard Atmosphere conditions at the latitude of 45°). An adjunctive attenuation due to rain for both horizontal and vertical polarization can be evaluated, but because of the raindrops shape (flattened on the bottom and with a curved dome on top) the one in the horizontal

polarization (shown in **Figure 9** in function of the frequency) is the most significant. Increasing the range, the rain attenuation is increasing too.

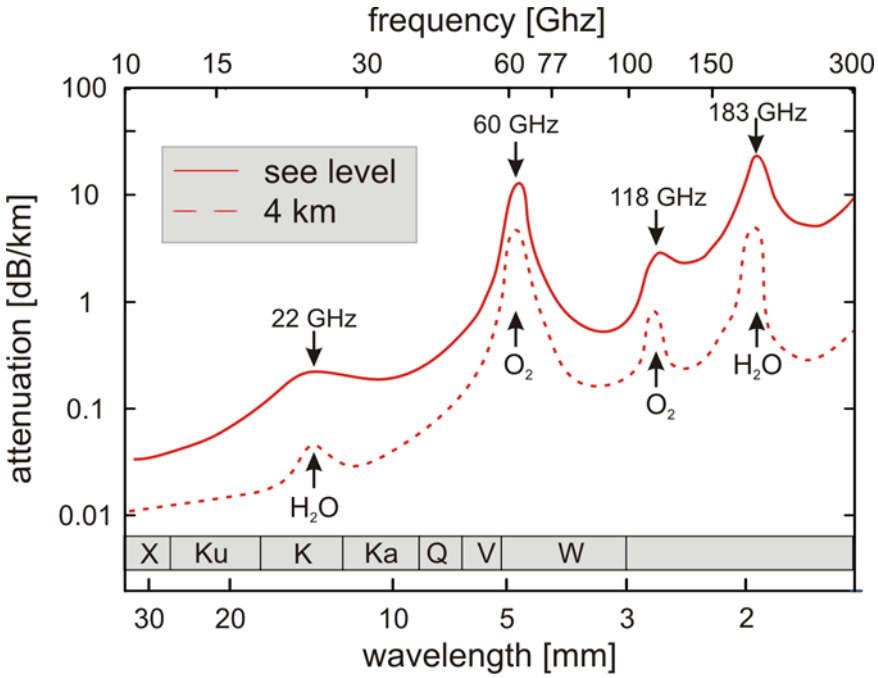


Figure 8 Atmospheric absorption curve.

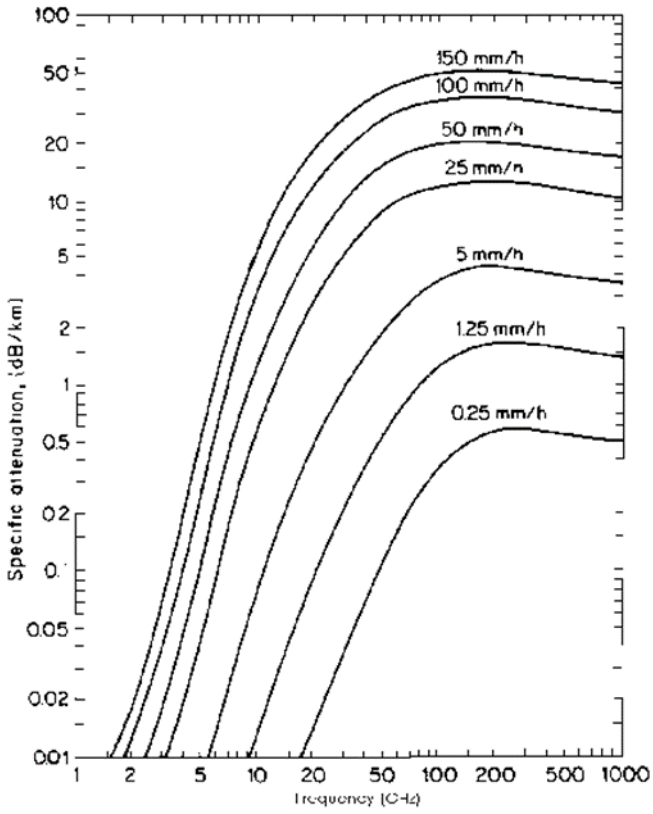


Figure 9. Speceific attenuation for different frequencies and rain rates

3.5 Rain rate evaluation

Generally, Z is also related to the rainfall intensity R [mm/h] according to a relation in the following form:

$$Z\,|_{linear} = a\,R^{b} \quad [\,mm^{6}m^{-3}] \tag{23}$$

where a and b can vary according to the precipitation type and R according to the rainfall rate. Therefore, the final procedure to evaluate the rain rate R, in function of the distance from the transmitter, starting from the scatterer received power measured from the radar is the following:

1) Compute $Konst$ on the basis of your operative conditions.

2) Choose a DSD and evaluate the coefficients a and b of Z-R relationship (23) according to the radar frequency you're using (Mie or Rayleigh theory).

3) Invert the equation (19) to extract Z from the received power:
$Z(dist)\,|_{dBZ} = \mathrm{Pr}(dist)\,|_{dBm} - Pt\,|_{dBm} - Konst\,|_{dB} + 180 + 20log_{10}(dist\,|_{Km})$

4) Invert the equation (23) to extract R from Z:

$$Z(dist)\,|_{linear} = 10^{\frac{Z(dist)|_{dBZ}}{10}} \tag{24}$$

$$R(dist) = \sqrt[b]{\frac{Z(dist)\,|_{linear}}{a}} \tag{25}$$

Obviously, the radar maximum range has been set before, to limit the reliable measurements according to the own choices. In this case, since the free space attenuation at 1 km is 130 dB, the 77 GHz radar is limited to, at most 100 m, thus it's a short range one.
There exists Z-R relationships with a and b already computed and considered standardized for each precipitation type, but they are valid within the Rayleigh's region.
Therefore, for my weather radar operating at 77 GHz, a proper Z-R relationship must be evaluated (see Pargraph 5.2).

Chapter 4

Hardware equipment

4.1 Radar sensor

The 77 GHz radar sensor I used for this thesis is: *RS3400W/04*, that can be controlled and powered with the *CO1000A/00* board, both produced by SilverSima. Sensor and board are connected through a cable and a standard circular horn antenna, belonging to Elva's SLHA series propagates the signal in the air. The following **Figure 10** shows the mentioned components:

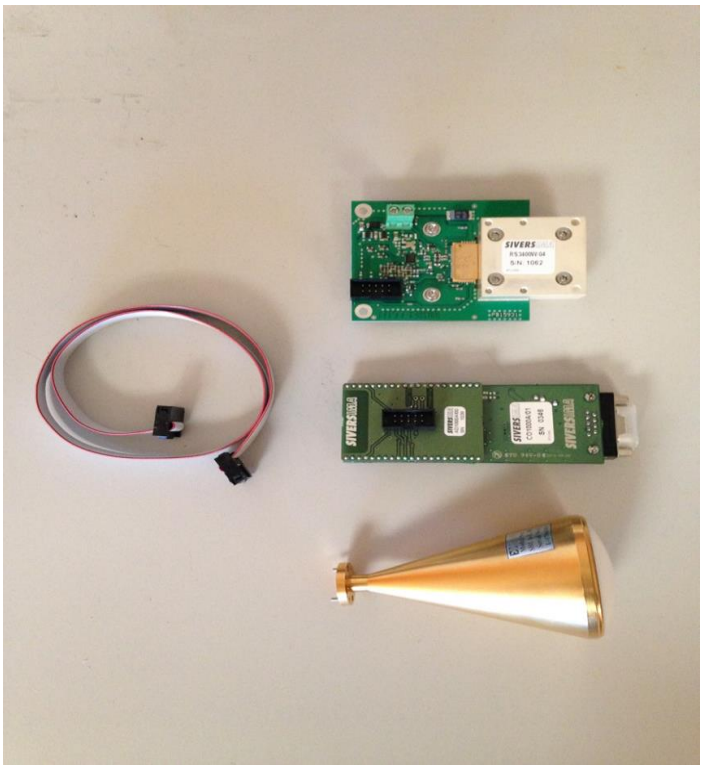


Figure 10. Radar sensor and antenna

I chose this particular radar sensor for different reasons:

1. It's easy to find on the market.
2. It has a relatively low cost.
3. It has small dimensions, so it is compact and maneuvrable.
4. It needs a low voltage power supply.
5. It can easily be controlled by PC, through a simple download of control interfaces.

The following **Table 1** shows some technical specifications of the radar sensor:

Table 1. Specifics of the radar sensor RS3400W/04

Typical Transmitted Power	4 dBm
Operational Frequencies	76 - 77 GHz
Chirp Bandiwidth (BW)	1 GHz
Transmitted Waveform	Sawtooth
Number Of Output Samples	1001
Max Output Signal Level	500 mVpp
Power Supply	5.5 V

4.2 Antenna

The circular horn antenna I chose, belonging to Elva's SLHA, is powered with the rectangular waveguide installed on the radar sensor boward and grants more than 25 dB of gain as shown by the following **Table 2**, together with other characteristics of the antenna:

Table 2. Antenna Parameters

Transmitter Gain (G_t)	28 dB
Receiver Gain (G_r)	28 dB
Length (L)	40 mm
Diameter (D)	28 mm
Return Loss	20 dB

4.3 PC

In order to control the radar sensor and develope my software, I also chose a very compact and silent (fan less) mini PC to be connected to the sensor itself through a USB-serial adapter. It was a good compromise between quality and price and its technical specifications are the following:

- Chassis: Black solid aluminum.
- Quad Core: Intel Celeron Processor J1900 (up to 2.42GHz).
- Ram: 2Gb.
- HardDisk: HDD – 500Gb.
- I/O Ports: 1xHDMI, 1xVGA, 2xLAN Ports (supporting Gigabit speeds), 4xRS232 (Serial port), 3xUSB 2.0 and 1xUSB 3.0.
- 2x Wi-Fi antennae.
- Default OS: Windows (32bit).

Figure 11. PC

In order to send manual commands to the radar sensor, an interface platform such as "*Termite*" can be used. Before starting the measurements, some parameters must be set:

- Hardware type (since this board can controll other radar sensors).
- Start frequency.
- Stop frequency.
- Sweep time.
- Sweep number.

Then the measurement mode must be turned on and the signal must be triggered (with other two commands). At this point, the acquisitions can start and the interface I chose automatically downloads the data acquired within a LOG file.
Sending again the trigger and the start commands will output the new measurements and this operation can be manually repeated whenever it's preferred.
Now it's more clear the aim the sofwtare I developed for this thesis: automate the measurement procedure in order to always have real time data, once the radar sensor has been fixed in a given position.

Chapter 5

Preliminary studies

5.1 Radar constant evaluation

In order to realize a fully operative 77 GHz radar prototype for rain estimation, I first had to check the project feasibility, performing some preliminary evaluations based on the mathematical derivations described in Chapter 3 and on other preliminary studies realized in the past years.

As illustrated in the pargraph 3.4, I am going to evaluate the radar constant on the basis of my operative conditions, considering the radar sensor and antenna technical specifications and supposing an additive attenuation of just 1 dB (since the max range is limited to up to 100m)

The following **Table 3** shows the values of the factors within the radar constant for the considered 77 GHz radar:

Table 3. Parameters for the evaluation of the radar constant

$\boldsymbol{\lambda}$	3.9 mm
$\boldsymbol{\theta_{3dB}}$	0.16 rad
$\boldsymbol{\varphi_{3dB}}$	0.16 rad
$\boldsymbol{G_r}$	28 dB
$\boldsymbol{G_t}$	28 dB
$\boldsymbol{L^2}$	1 dB
$\boldsymbol{K^2}$	0.75
$\boldsymbol{BW}$	1 GHz

Now it is possible to calculate the radar constant in dB:

$$Konst\,|_{dB} = 10\,log_{10}(c) + 30log_{10}(\pi) + G_r + G_t + 10log_{10}(\theta_{3dB}) + 10log_{10}(\varphi_{3dB}) + 20log_{10}(K) + L - 10log_{10}(BW) - 20log_{10}(\lambda) - 10log_{10}(1024(ln2))$$

Substituing the numbers from the **Table 3**:

$$Konst\,|_{dB} = 69.15$$

5.2 Z-R Derivation

Now, a proper Z-R relation at 77 GHz must be found out. The procedure is strictly empirical, even if based on the theoretical considerations I largely discussed within chapter 3 and consists of the following steps:

1) Consider the Marshall and Palmer DSD:
2) Consider the Z expression of Mie.
3) Consider a large set of rainfall rates R and assume that the Z-R relationship has the form of equation (23).
4) Evaluate the values of Z through equation (20) considering all the rates in the set, that influence the DSD and so the integral computation.
5) Linearize the (23) as follows:

$$Log(Z) = Log(a) + b\ Log(R)$$

6) Compute the values of the coefficients a and b using the least squares method.

Using a Matlab code to execute these steps, the following Z-R relationship has been found out for the 77 GHz radar and the standard conditiond included in the Marshall and Palmer DSD:

$$Z = 119\ R^{0.67}$$

The following **Figure 12** shows a comparison between Mie and Rayleigh (using Marshall and Palmer standard coefficients) models in standard DSD conditions, pointing out the R (represented in logarithmic scale) underestimation by Rayleigh, at this frequency, having fixed a value of Z (overestimated):

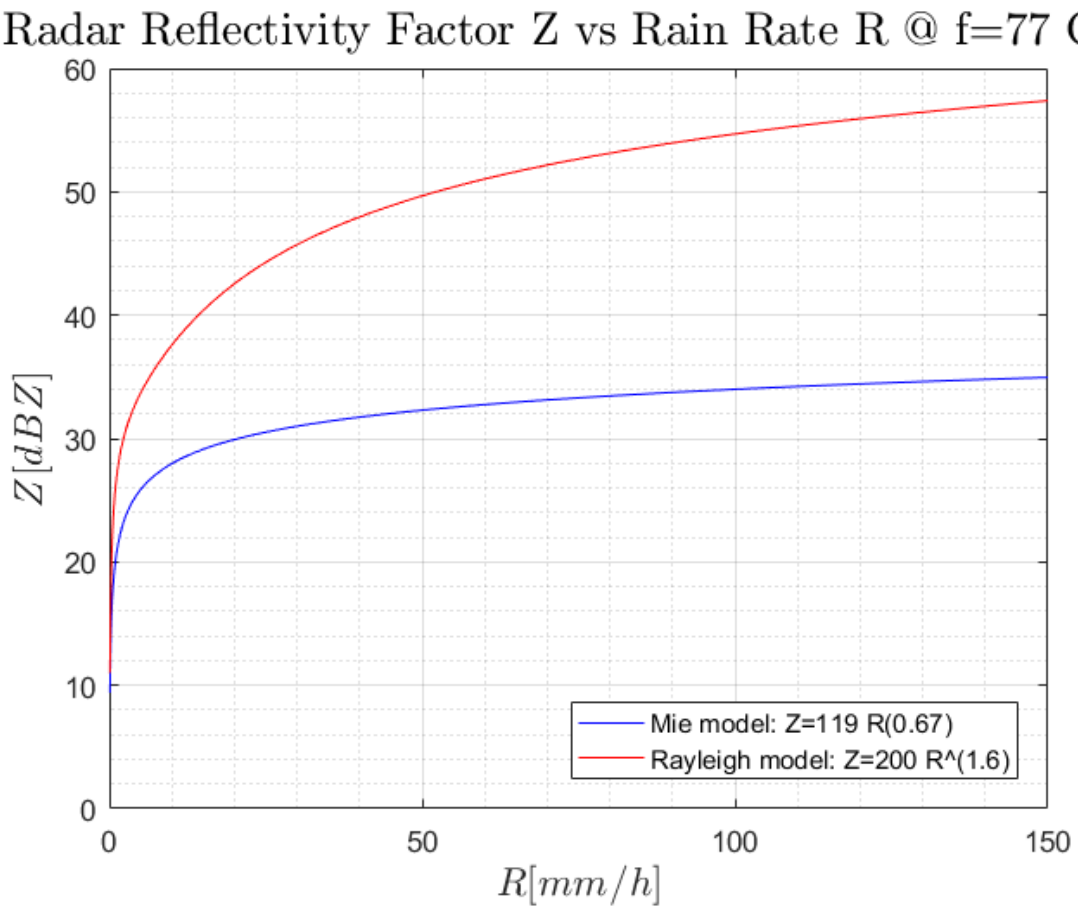


Figure 12. Mie Vs Rayleigh (standard DSD)

Changing the DSD with the one presented by Joss for thunderstorm and running again Matlab code mentioned above, the following Z-R relationship for the 77 GHz radar comes out:

$$Z = 67\, R^{0.59}$$

The following **Figure 13** shows a new comparison between Mie and Rayleigh models: the *R* underestimation of Rayleigh is still evident:

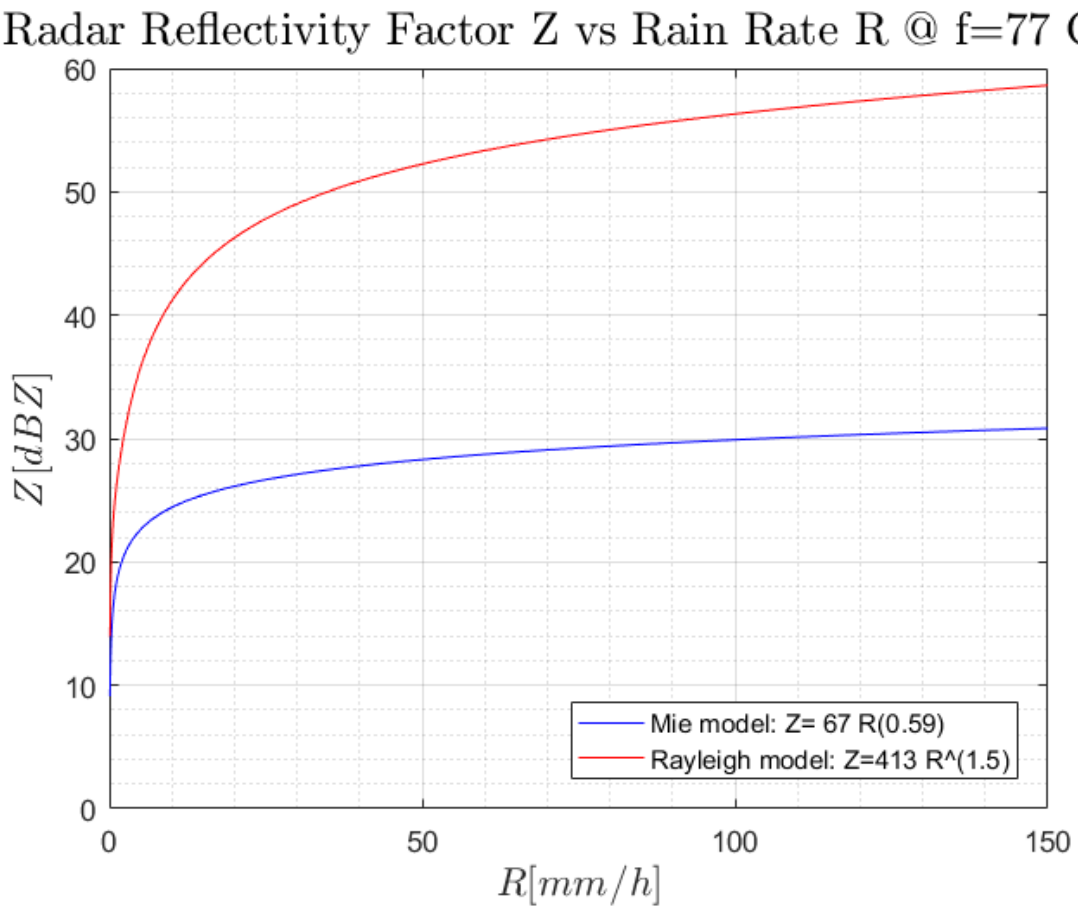


Figure 13. Mie Vs Rayleigh (Joss DSD for thunderstorm)

The following **Table 4** shows some correspondances between $Z\,|_{dBZ}$ and R values in standard DSD conditions ($Z = 119\ R^{0.67}$):

Table 4. Rainfall rate and dBZ

Rain Type	dBZ	Rainfall rate [mm/h]
light to moderate	14-24.5	0.1-5
moderate to heavy	24.5-29.5	5-20
heavy to very heavy	29.5-33	20-70
very heavy to intense	33-35	70-130
extreme	> 35	More than 130

5.3 Acquisition and processing parameters

Once the project feasibility has been checked, the radar sensor must be configured. Knowing that the starting and the stopping frequency are respectively 76 GHz and 77 GHz (as reported in **Table 1**), just two parameters need to be set:

- Sweep Time → 75 ms (this is also the default value).
- Sweep Number → 10.

This means that the frequency sweep is performed 10 times (each one lasting 75 ms) within a measurement session and outcome IF signal values (*Data*) are averaged over 10.
These signal values returned from the radar are reported in digital units. So, during the processing operations, in order to obtain the real power measured by the radar, the following procedure (reported below here as pseudocode) must be followed:

1. $Data_{filt} = Data - mean\,(Data)$.
2. $Data_{Volt} = \frac{Data_{filt}}{2000}$.
3. $Data_{Power} = \frac{(Data_{Volt}^2)}{10000}$.

The data filtering, with the mean value, is needed to remove the continuous component of the signal. Then, to convert the digital units in the maximum output signal level, again specified in **Table 1,** of 500 mV, they have to be divided by a factor 2: infact, two digital units are equal to 1 mV. The conversion in Volt is done with the division for a factor 1000 (complexively 2000).
To finally obtain the power, the classical formula $\frac{V^2}{R}$ is applied, considering $R = 10\ kOhm$, as specified in the application notes of the radar sensor.
In order to discover the correspondence between frequency and power peaks, corresponding to targets individuated by the radar sensor, the Fourier Transform has to be applied to the power signal just computed.
Knowing that the sensor returns 1001 samples, it's possible to compute its A/D converter sampling frequency:

$$F_s = \frac{nSamples}{T} = 13.347\ kHz \tag{26}$$

Being the number of FFT samples $nFFT = 1024$, since it's the next power of 2 closest to the number of samples, the FFT resolution can be computed:

$$Res_{FFT} = \frac{F_s}{nFFT} = 13\ Hz \tag{27}$$

Converting this frequency in distance, substituting in the equation (4) of the target distance, the frequency with the FFT resolution, it turns out that the Fourier transform, in this case, doesn't degrade the radar performances. Its resolution in meters is:

$$Res_{FFT}\,|_m = \frac{c\ T\ Res_{FFT}}{2\ BW} = 0.146\ m \tag{28}$$

And this value is infact a bit lower than the radar resolution computed with (5).
The maximum achievable distance with this parameter setup, considering that the Fourier Transform is symmetric with the respect to the origin, and so that just half of the samples must be kept in account, is:

$$R_{max} = \frac{c\ T \frac{F_s}{2}}{2\ BW} = 75\ m \tag{29}$$

In such a way, I am going to widely respect the 100 m limit supposed in paragraph 3.5.
The following **Table 5** summarizes the above mentioned parameters:

Table 5. Acquisition and Processing parameters

$\boldsymbol{T}$	75 ms
Sweep Number	10
$\boldsymbol{F_s}$	13.347 kHz
$\boldsymbol{Res_{FFT}}$	13 Hz
$\boldsymbol{Res_{FFT}\,|_m}$	0.146 m
$\boldsymbol{R_{max}}$	75 m

5.4 Preliminary test

5.4.1 Interface configuration

At this point, the radar configuration seen in the previous paragraph has to be applied.
I chose, for simplicity, "*Termite*" interface, as already mentioned before, to first set up the serial communication (between my computer and the electronic board of the radar sensor) in such a way:

- Baud Rate (number of symbols transmitted per second): 115200.
- Data Bits: 8.
- Stop Bits (to signal the end of every character): 1.
- Parity Bits (to detect errors): none.
- Control Bits (to control the data stream): none.
- CR-LF (Carriage Return – Line Feed): yes.

Once the interface detects the presence of the electronic board, I can send it the commands, by simply writing them and pressing "*ENTER*", to configure the sensor and start the measurements. The board will reply with " *OK* " if the command has been correctly received, otherwise with " *?* ".
The necessary commands to begin the measurements are the following (the order should be respected, to avoid errors, exspecially for the last two commands that actually start the signal transmission), as already briefly mentioned in paragraph 4.3:

1. INIT
2. HARD:SYST RS3400W
3. FREQ:START 76e09
4. FREQ:STOP 77e09
5. SWEEP:NUMBERS 10
6. SWEEP:TIME 75e-3
7. SWEEP:MEAS ON
8. TRIG:ARM
9. TRACE:DATA ?

Commands 8 and 9 have to be sent, after the ending of the measurement cycle, to repeat it again. The command 9, in particular, let to download the data buffered in the board.

5.4.2 Outdoor obstacle detection test

Before testing the 77 GHz radar sensor as rain estimator, I preferred to check if it was working correctly as an obstacle detector.
So, I went over the Politecnico's rooftop and I mounted the equipment (I used my laptop rather than the mini pc I chose, since it needs a monitor and it was complicated to take on the rooftop):

Figure 14. Outdoor test: setup

In **Figure 14**, I show some of the equipment, to give an idea of the setup.
For the concerns of the surrounding environment, the following **Figure 15** shows the location where I wanted to test the radar as obstacle detector:

Figure 15. Outdoor test: environment

The circled objects in the image are the objects I'd like to detect.

For the data processing procedure, I illustrated in the previous paragraph 5.3, I wrote a Matlab code outputting different plots, including the two following ones:

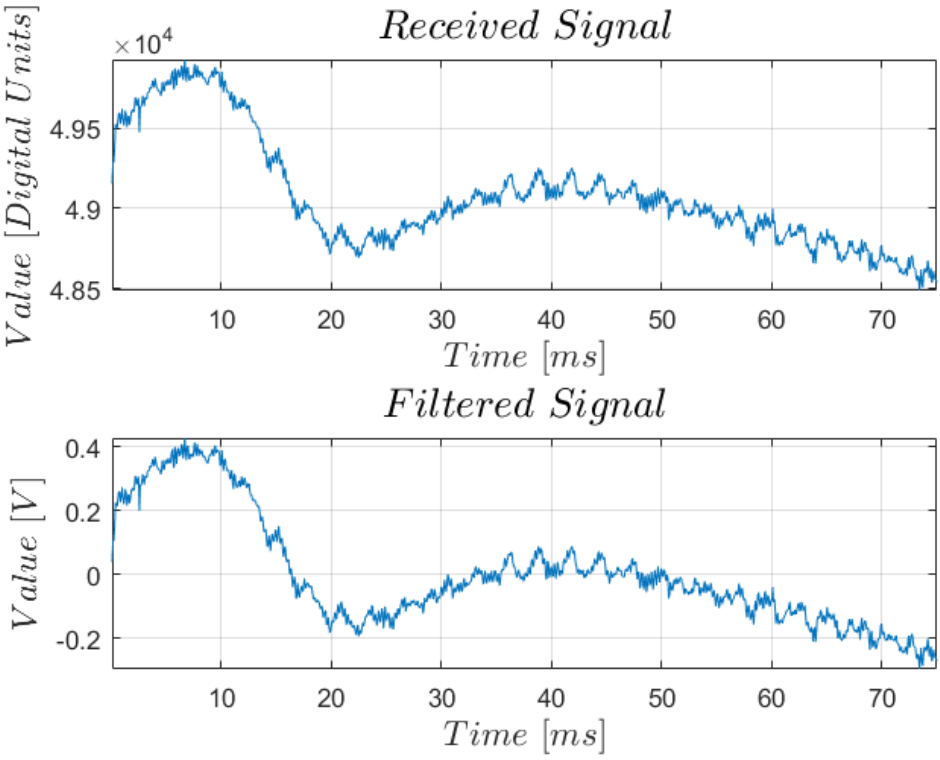


Figure 16. Outdoor test: output signal

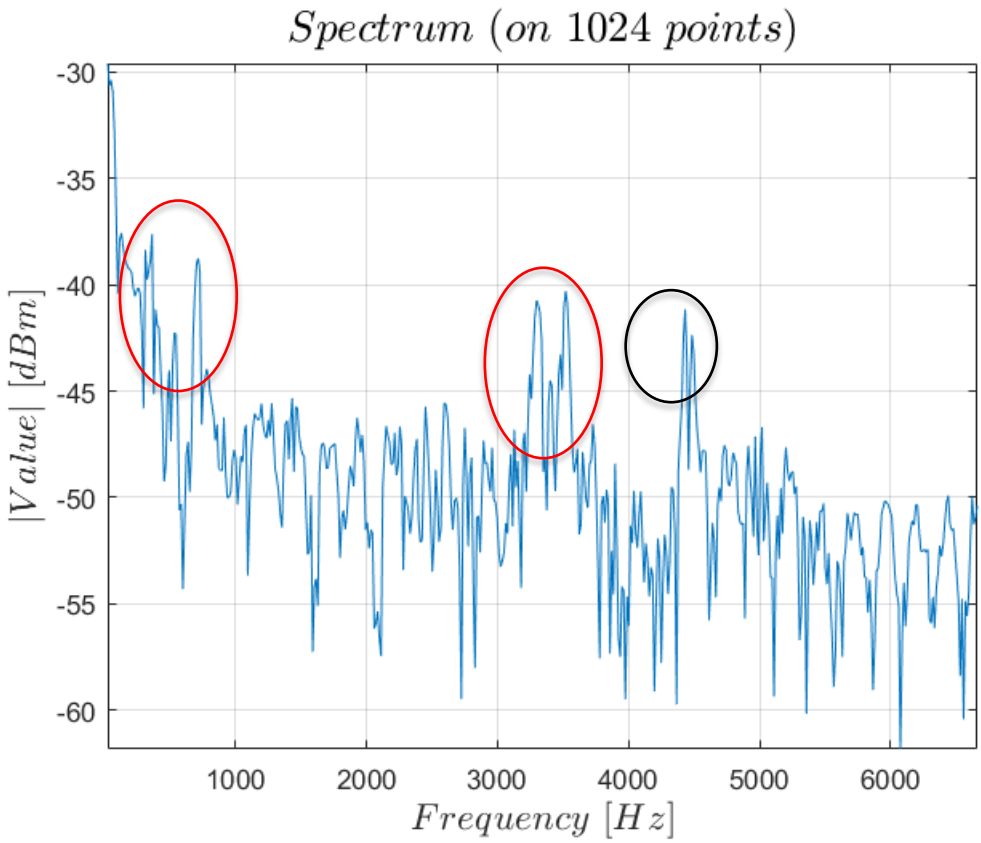


Figure 17. Outdoor test: output signal spectrum

The **Figure 16** shows, on the top subplot, the IF output signal, of the radar sensor, evolving within the sweep time, after its values have been averaged for the 10 sweeps (essentially it's the $Data$ I mentioned in paragraph 5.3). On the bottom subplot, instead the same signal, converted in Volt is represented (essentially $Data_{Volt}$ in paragraph 5.3) and it can be noticed how effectively its a 500 mVpp signal, as the specifics report (**Table 1**).

The **Figure 17** simply gives the spectrum (just the positive frequencies, obviously, since it's symmetric) of the $Data_{Power}$ signal (obtained from $Data_{Volt}$ after the transformations discussed in paragraph 5.3). The red-circled peaks within the plot are reported below, with the corresponding distances computed with equation (4):

- 312.81 Hz → 3.52 m
- 364.95 Hz → 4.10 m
- 716.86 Hz → 8.06 m
- 3297.6 Hz → 37.09 m
- 3519.1 Hz → 39.88 m

The distances within the 10 m range refer to the two metal bars in the foreground, while the other refer to the air conditioner in background of the **Figure 15**.

Since I also checked the obstacles distance from the radar sensor also with a yardstick, comparing the measurements with the ones obtained from the power spetrcum, I can validate this preliminary test: the radar sensor is correctly able to detect target distances.
The black-circled frequency peaks (at 4431.5 Hz and 4483.6 Hz), instead, will be a constant presence of all the measurements, so I checked and verified that is due to the electronics (antenna, radar sensor or power supplier).

5.4.3 Radar calibration and rain estimation test

After some experiments with the 77 GHz radar as rain gauge, I found that it needed calibration, due to an internal controller responsible for an excessive distance compensation (increasing too much the Z values), since this type of radar was designed to cover long-range distances. The calibration simply consists in an additive modification of the Z values obtained after the equation (19) inversion:

$$Z(dist)\,|_{dBZ} = Z(dist)\,|_{dBZ} - 10log_{10}(dist\,|_{Km}) - \ 20 \tag{30}$$

This calibration is valid only for this specific radar sensor and after that, the passage to linear values can be done by simply applying the equation (24).
Since the Matlab code I wrote is unique and includes also the Z calculation and the R one, the correction exposed with equation (30) was applied also to obtain the results of the previous paragraph. However, the Z values were not important for target detection, so I mentioned it just now.
The results I am going to show are relative to one of the experiments made to prove that this 77 GHz radar can be used also as rain gauge.
It was November 6th, a rainy day and I was in an office, in Politecnico. I mounted the equipment as shown in the following **Figure 18**, picking up the mini PC I chose, instead of the my laptop, to check the complete prototype functionality:

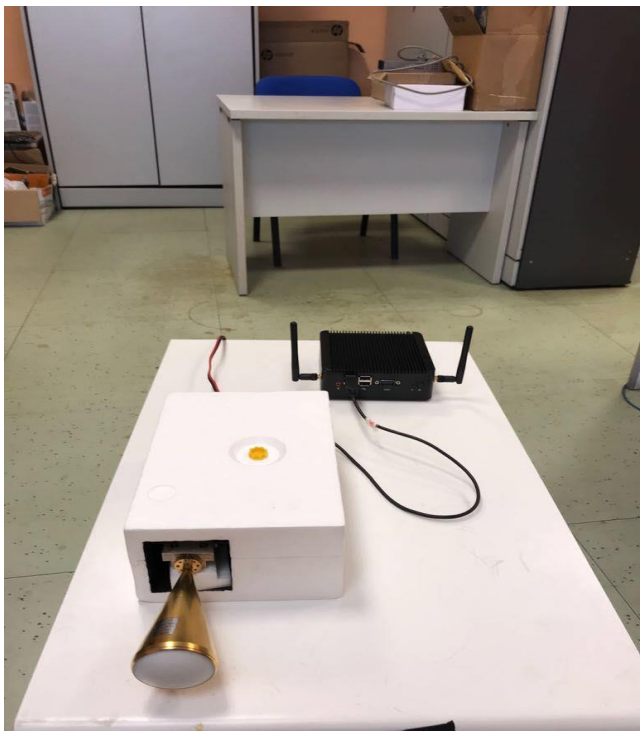


Figure 18. Rain gauge test: setup

The radar was pointed outside the window office, whose environment is displayed in the following **Figure 19**:

Figure 19. Rain gauge test: environment

I started the acquisition procedure at different times and I chose one of the $Data$ outputted from the radar sensor, as input for the above mentioned Matlab code I wrote. The results are reported below:

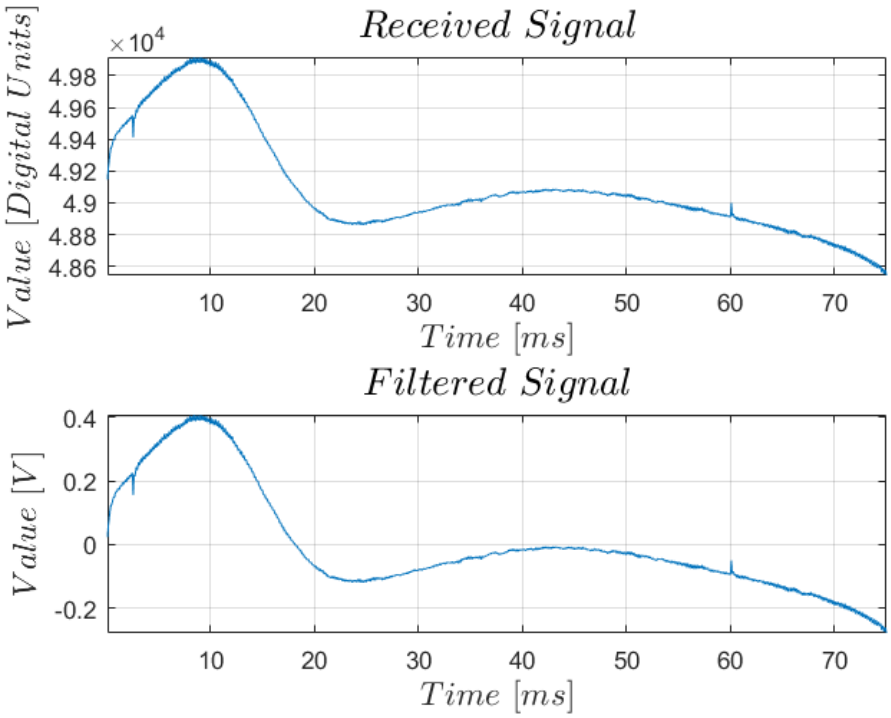


Figure 20. Rain gauge test: output signal

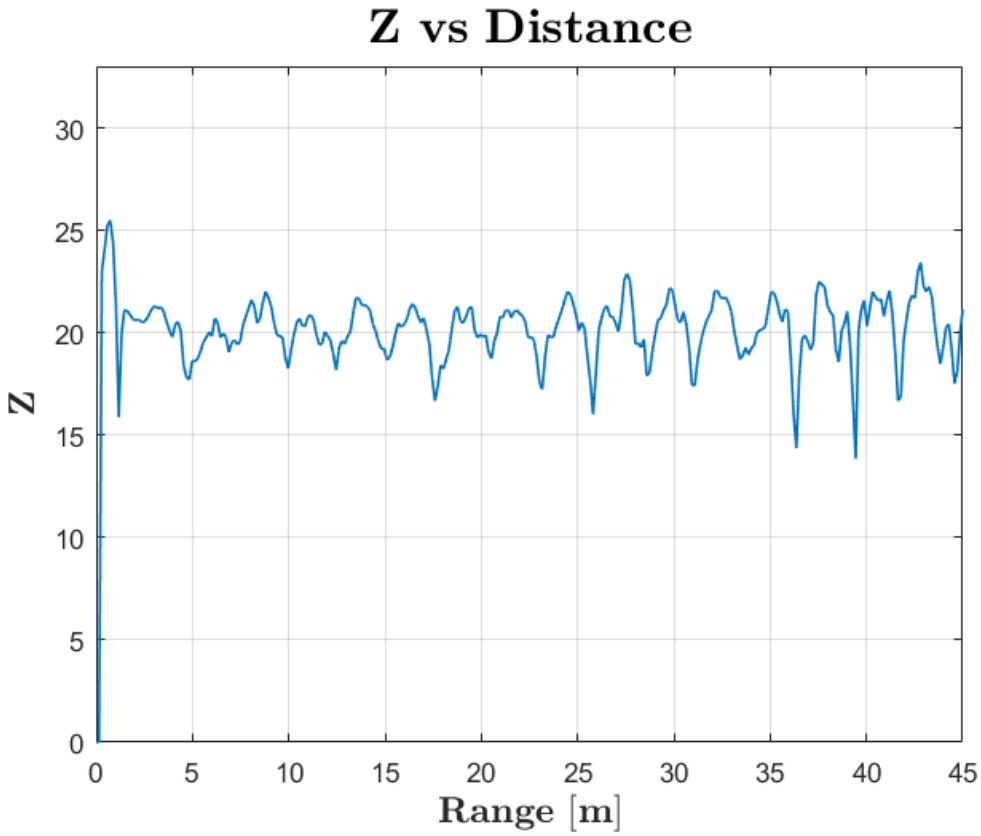


Figure 21. Rain gauge test: output Z values

From **Figure 20,** it can be noticed that the signal, expressed in Volt, in the bottom subplot, is again respecting the 500mVpp limitations and that it is less noisy than the one outputted by the outdoor test (**Figure 16**). This is reasonable, since the environment was completely different, richer of obastacles and reflecting surfaces close to the radar sensor. In this case, as the **Figure 18** shows, the radar sensor view is characterized by a more open environment and the main reflections come from the rain drops.

Infact, **Figure 21** reports oscillating Z values around 20 dBZ, comparable with the light rain rate of that moment (see **Figure 12** and **Table 4**). The range has been limited to 45 meters due to clutters and interferences, that would have disturbed the rain rate computation, appearing higher. The initial peak is probably due to reflections caused by the window, since the sensor hasn't been completely placed outside it.

Having chosen as Z-R relationship the one exposed in paragraph 5.2 for standard DSD ($Z = 119\ R^{0.67}$), the rain rate results plotted in the following **Figure 22** confirm the validity of the measurements:

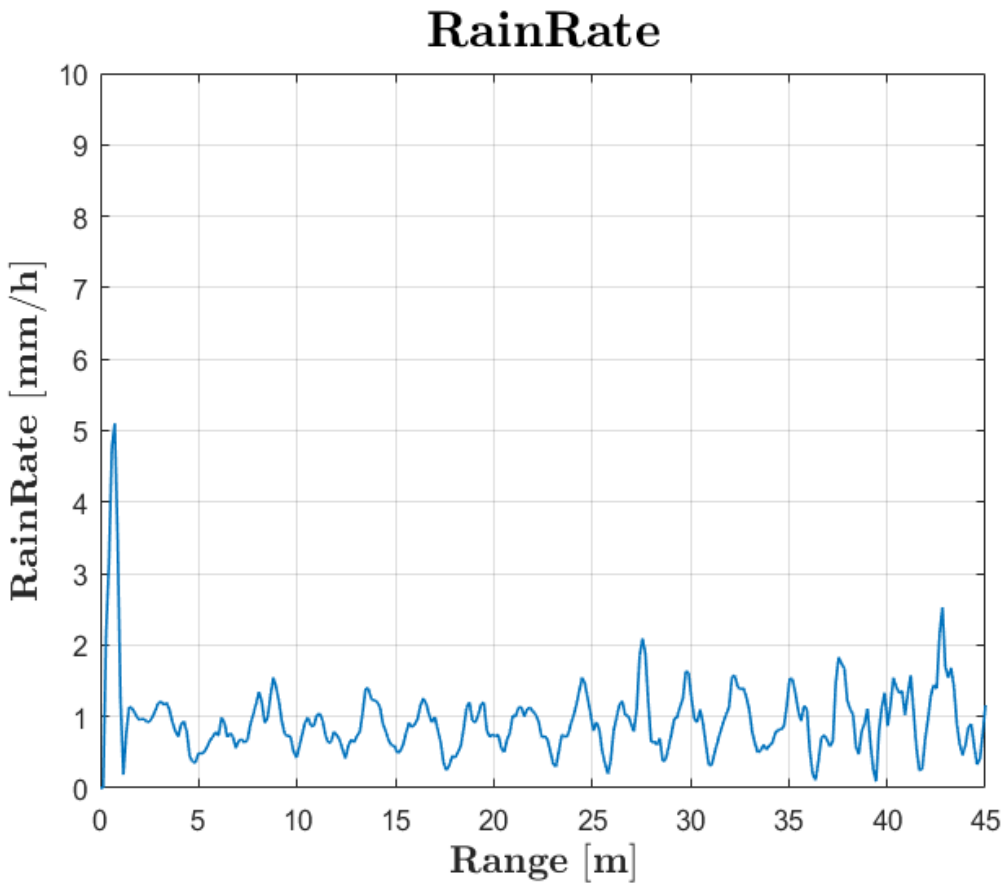


Figure 22. Rain gauge test: rain rate

The initial peak derives from the same peak of Z values reported above. The rain rate seems to be quite constant and on average equal to 1.5 mm/h in the operativity range, thus coherent with the light rain registered at the instants of measurement.

Chapter 6

Software development

6.1 PC Configuration

Since the prototype is fully operative, I can start developing the software on the mini PC. First of all, I needed to configure it and I chose to partition the hard disk, mounting a Unix Operating System: Ubuntu 16.04. In such a way, due to Linux flexibility, being it as an open source, it has been much easier to configure it according to my needs and also to manage the internal settings. Then I decided to write the software in C-language and I installed Eclipse as development environment. Through the terminal, just a few lines have been enough to properly set everything, including the Java internal updates.
Before illustrating the software structure I designed, I am going to explain how I also had to configure the kernel for the serial communication (the connection between radar sensor and mini PC is realized with the serial-USB adapter).
First, I checked the supports present in my mini PC using this command:

```
dmesg | grep tty
```

Then, I chose one of the USB ports physically available (`ttyUSB0`), which Linux interpret as a text file. Knowing that, I will design my software in such a way that the port will be opened, closed, read and written, as a simple file. But the permissions are needed, thus I used the two following commands:

```
sudo chmod 777 ttyUSB0
sudo adduser luca dialout
```

Finally, the user I created on Linux belongs to the group `dialout` owner of the serial port, thus Eclipse is enabled to access it.
Having done these preliminary configurations, I started the software development, that I am going to illustrate in the following paragraphs, pointing out the software structure and then analyzing the single functions and libraries.

6.2 Software structure

My software is structurated in 15 files, both source (.c) and header (.h) ones, including different functions and libraries I created to execute various operations. They are listed below and their details arepresentated in the following paragraph:

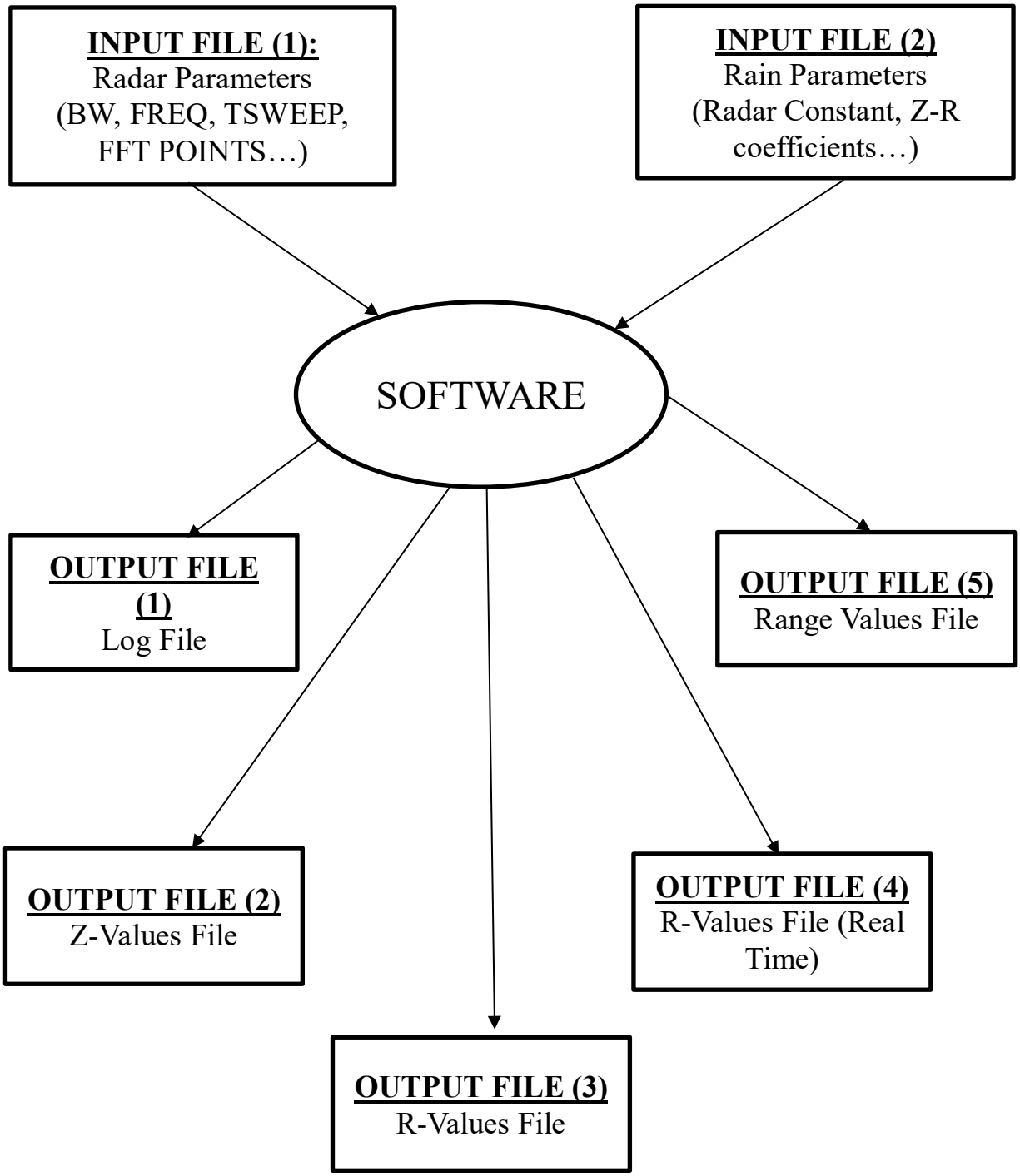
- *core.c*
- *myglobal.h*
- *myinclude.h*
- *debug.h*
- *logManager.c*
- *logManager.h*
- *xmlManager.c*
- *xmlManager.h*
- *serial.c*
- *serial.h*
- *acquisition.c*
- *acquisition.h*
- *processing.c*
- *processing.h*
- *myerror.h*

The header files, having a name equal to a source file, are just used to point out the functions prototytpe used in the corresponding source file. The remaining header files, instead, contain global variables, libraries called within the whole software, debug functionalities and error issues. Everything works through the set of all these files and the cooperations of all the functions and variables I defined.
The software, generally speaking, takes two XML files as input (*"acqPar.xml"* and "*procPar.xml"*), containing, respectively, parameters related to the radar sensor and to the Z-R relationship computation, and produces five files types as output:

1. A log file, reporting everything the software does while it is running.
2. A Z-File containing the Z values averaged for all the acquisitions performed in the last minute.
3. An R-File, similar to the Z-File, containing the rain rate values of the last minute.

4. A real time file containing real time rain rate values.
5. A file containing the range values considered for the Z-R computation.

The following block diagram shows the software IN/OUT scheme:



Except for the Log File, updated each time an action within the software has to be registered, and for the Range values files, created just once (its utility will be clarified in Chapter 7, where the web site development, for data presentation is illustrated), the other files are outputted every minute. In particular, a Z-File and an R-File are created by new, while the Real Time file is updated with the values of the last R-File created (actually the utility of this last file will be clarified in Chapter 7).

6.3 Software analysis

At this point, I am going to focus singularly on each file mentioned above.

6.3.1 core.c

This file is the actual core of the software, since it contains the "main", the code immediately executed when the program starts. It is a very minimal file, including only calls to functions that I have implemented in the various libraries I created.
In particular, these functions are needed to perform the following steps within the software:

1. Opening of the Log file with registration of its starting time.
2. Opening of the communication with the serial port.
3. Setting of the radar measurement parameters, read from the corresponding XML file.
4. Testing of the acquisition to check if everything works and to save the number of samples received by the radar.
5. Loading of the parameters for the processing from the corresponding XML file.
6. Starting of the acquisitions (their number per minute can be chosen from the acquisition parameters XML file) loop.
7. Eventual closing of the communication with the serial port and of the Log file (I used these functions at the end just to test the software; actually it has to work indefinitely, thus they will never be called).

Before running the program, thus this main file, the serial communication must have been properly set as illustrated in the previous paragraph, and also the XML library must be configured too (see paragraph 6.3.6).

6.3.2 myglobal.h

This file contains both the #define directives for the compiler and the declaration of all the global variables I used within the software.
The compiler directives can be essentialy grouped in three subsets:

1. Absolute constant values: the squared pi-greco (*PI2*) and the max unambiguous range chosen for the radar.
2. Pathname for the Input and Output files and for the serial port.
3. Commands to be sent consecutively to the serial port, to configure the acquisition process (they are specified in paragraph 5.4.1).

For what concerns the global variables, I used them since different functions I implemented may have needs for them. For example, they include all the file pointers, the buffer to save each string returned from the radar sensor, and a data structure to save all the processing parameters read by the corresponding XML file (Z-R relationship coefficients, the radar constant, the transmitted power, ecc…).

6.3.3 myinclude.h

This file contains both the header files of the libraries I created and the ones of the standard public libraries such as:

- *stdio.h* (containing standard input/output definitions).
- *stdlib.h* (containing general utility definitions).
- *string.h* (containing utilities for string manipulation definitions).
- *errno.h* (containing error definitions).
- *termios.h* (containing terminal I/O definitions).
- *signal.h* (containing signalling constant definitions).
- *time.h* (containing utilities for time manipulation definitions).
- *math.h* (containing mathematical operation definitions).
- *libxml/tree.h* (containing utilities for XML manipulation definitions).
- *libxml/xmlreader.h* (containing utilities for XML manipulation definitions).
- *libxml/parser.h* (containing utilities for XML manipulation definitions).
- *sys/types.h* (containing general utilities definitions).

- *sys/ioctl.h* (containing utilities for I/O interaction definitions).

All the source files contain the #include for this header file, in such a way the code are cleaner, with respect to the repletion of all the libraries includes.

6.3.4 debug.h

This file implements the eventual activation of all the debug sections I put within the whole software. They essentially consist in pieces of code that helped me a lot during the debug phase of the software, to check for errors and if in general everything worked fine. I considered five types of possible debug sections, each more or less included in its corresponding source file:

1. DEBUG_LOG to check if what has been written in the Log file is correct.
2. DEBUG_SERIAL to check if what has been written in the serial port or what has been read from it, is correct and consistent with the commands sent.
3. DEBUG_XML to check if the root of the XML document and the content of the current text node are correct.
4. DEBUG_ACQ to check if the time step between consecutive acquisitions correspond to the one chosen before and if the number of samples returned from the radar is consistent with the theoretical value of 1001.
5. DEBUG_PROC to check if the following quantities are consistent with what has been chosen or expected: the number of FFT points (read from the "*procPar.xml"* file), the content of the frequency and range vectors, used for processing, and the power spectrum values in dB and dBm respectively.

In order to check all these elements, they are simply printed out by the software if the corresponding debug section has been activated. This happens if its value in "*debug.h"* has been set to 1. Otherwise, the content of the debug sections within the different source files, is ignored during the execution.

6.3.5 logManager.c

This file contains the implementation of the following functions, used to manage not only the Log file, as in origin was thought for, but all the software output files:

1. *FILE * **open_log** (char * filename)*
2. *int **write_log** (FILE * fp, char * buf)*
3. *int **close_log** (FILE * fp)*
4. *void **timing_log** (char * buf)*

The first function is essentially used for the file opening: it takes, as parameters, the filename, including the pathname, and opens the file through a file pointer that is then returned. There is a simple distinction between the Range and the Real time value files, and all the others: the first ones are opened in writing mode, since each time they are opened with eventually different and consecutive runs of the software, they need to be refreshed; all the other files are opened in append-mode because, for instance, the Log file needs to be just updated, and not completely refreshed, each time something occurs within the software.
The second function is instead used for the file writing: it takes, as parameters, the file pointer and a string containing the characters to be written in the file, and the number of charcaters actually written in then returned by the function. In order to empty the buffer of the *fprintf* function that is called for the writing, and let the file updating in real-time, the *fflush* function is also adopted.
The third function is used to close the file referenciated by the file pointer taken as parameter.
The fourth function, finally, reports the time reference for a given action for which it is called: it builds a string, concatenating the one takes as parameter, with the actual local time, and writes it on the file.

6.3.6 xmlManager.c

This file contains the implementation of the functions I needed to manage the XML files, taken as input from the software. Since I made use of the *libxml2* library, I'll briefly show how I installed it on Ubuntu of the mini pc and then how I added it to Eclipse.
First, I downloaded the last version of the library from the *xmlsoft.org* archive by using the following command from terminal:

```
wget ftp://xmlsoft.org/libxml2/libxml2-sources-version.tar.gz
```

Then, I extracted files from the downloaded package and I entered the directory where it was extracted:

```
tar -xvzf libxml2-sources-version.tar.gz
cd libxml2-version
```

At this point, I configured and compiled the library, choosing the directory path where I wanted to copy files and folders:

```
./configure --prefix=/usr/local/libxml2
make
```

Finally, I installed the library as super user, due to the destination directory privileges:

```
sudo make install
```

To check if everything had properly worked, I explored the library content, and within the directory */usr/local/libxml2/libxml* I found all the header files, referring to the functions already implemented in the library, and I took some of them, according to the developing needs.
In particular, as I already mentioned in *myinclude.h* (paragraph 6.3.3), I just needed of: *libxml/tree.h, libxml/xmlreader.h* and *libxml/parser.h*.
In order to let Eclipse to access *libxml2* library, in my software environment, I had to modify the project properties and in particular:

- In C/C++ Build → Setting I had to include the library path for the GCC compiler and the library name for the GCC linker.
- In C/C++ General → Paths and Symbols I had to again include the library path.

Now, I am going to show how the two input XML files are organized. They both contain a first line indicating the XML version I used and the encoding (here *UTF-8*), and a second line indicating the document root.
For what concerns *"acqPar.xml"*, the root is <*AcquisitionParameters*> and all the children nodes are text ones, having as values respectively the number of acquisitions per minute to be performed (according to the proper needs) and the values used within the commands to be sent to the serial port (specified in paragraph 5.4.1). In particular, the children nodes I used in this file are:

- <*HardwareType*> `value`</*HardwareType*>
- <*FrequencyStart*> `value`</*FrequencyStart*>

- <*FrequencyStop*> `value`<*/FrequencyStop*>
- <*SweepNumber*> `value`<*/SweepNumber*>
- <*SweepTime*> `value`<*/SweepTime*>
- <*AcqPerMinute*> `value`<*/AcqPerMinute*>

For what, instead, concerns *"ProcPar.xml"*, the root is <*ProcessingParameters*> and all the children nodes are text ones, having the values of some parameters to be used for Z-R computation, and general processing. In particular, the children nodes I used in this file are:

- <*RadarConstant*> `value`<*/RadarConstant*>
- <*txPower*> `value`<*/txPower*>
- <*BW*> `value`<*/BW*>
- <*SweepTime*> `value`<*/SweepTime*>
- <*LightSpeed*> `value`<*/LightSpeed*>
- <*a*> `value`<*/a*>
- <*b*> `value`<*/b*>
- <*MinDistance*> `value`<*/MinDistance*>

At this point, I am going to illustrate how I implemented the two functions within the source file presented in this paragraph (*xmlManager.c*):

1. *xmlNodePtr* ***xml_open*** *(char * filename)*
2. *char* ***** ***xml_readTextNode*** *(xmlNodePtr cur)*

The first function is essentially used for the XML file opening: it takes, as parameter, the filename, including the pathname, and parse it, creating a tree data structure containg all the document nodes. Then, the root of the document is returned as *xmlNodePtr* type, as specified by the *libxml2* I adopted.
The second function is instead used for the single text node reading: it takes, as parameter, the *xmlNodePtr* referred to the text node to (of the XML document previously parsed) to be currently read. The content of the text node, is then returned as a string.

6.3.7 serial.c

This file contains the implementation of the following functions, used to set correctly the serial communication with the radar control board and to properly manage it:

1. *void* ***open_port*** *(char * filename)*
2. *void* ***close_port*** *(FILE * fp)*
3. *void* ***write_port*** *(FILE * fp, char * wrBuf)*
4. *char * **read_port** (FILE * fp)*
5. *void* ***serial_setup*** *(FILE * fp)*

The first function is essentially used for the serial port opening: it takes, as parameter, the filename, including the pathname, that Ubuntu has assigned to the serial port (for example */dev/ttyUSB0* identifying the USB-serial adapter I chose, as already explained in paragraph 6.1). Then, the port is opened as a file (in the corresponding "append" mode of a file, to grant its buffer updating (both for reading and writing), and since the file pointer is a global variable (declared in *myglobal.h*), this function doesn't return anything. In order to correctly setup the serial communication, I used the *termios* global data structure that contains the port options control. In particular, for the communication with the radar control board, I set the following parameters values, as explained within the radar application notes:

- Input BitRate: 115200.
- Output Bitrate: 115200.
- CS8 Configuration: 8 Data Bit, No Parity, 1 Stop Bit.
- Flow Control: Disabled.
- Modem Controls: Disabled.
- I/O Carriage Return New Line: Enabled (the carriage return character is interpretated as the beginning of a new line).

After having checked for errors, before returning, the function flushes the Input buffer of the serial port, to avoid the data overwriting.
The second function is instead used for the serial port closing: it takes, as parameter, the file pointer referred to the port and closes it, checking for errors and writing the action on the Log file, as for the greatest part of operation performed within the software.
The third function is used to send characters to the serial port, thus to write on it: it takes, as parameters, the file pointer referred to the port and the string containing the data to be sent.

Then, they are written, as for a simple file and the output buffer of the serial port is flushed, to ensure the data reception in real-time.

The fourth function, in a similar way, is used to read characters from the serial port: it takes, as parameters, the file pointer referred to the port and returns the string containing the data read, that will be processed. This string has a statically pre-allocated dimension, since this is the simplest way to return string from functions in C, but this doesn't create any problems: the controller board, indeed, just returns "*OK*", "*?*" or a number (sample from a measurement), as already explained in paragraph 5.4.1.

The fifth and last function is an essential one, since it initializes the radar acquisition parameters: it takes as parameter just the file pointer referred to the port, exploits the predefined functions of the *libxml2* library and the ones I implemented in *xmlManager.c*, and doesn't return anything.

First of all, the "*acqPar.xml* " file is opened and all the parameter values it contains (specified in the previous paragraph) are saved in a strings array. Then, through a string concatenation (with the #define specified in *myglobal.h*), the nine different commands to be consecutively sent to the serial port are built up. After that, there is a rapid check on the radar model: if it coincides with *RS3400W*, the *flag77* variable is set to the value 1, otherwise remains equal to 0. This distinction is an example of how this software is versatile and can be used with different radar model sharing the same electronic control board. However, after sending each command, thus writing it on the serial port, the functions checks if it has been received correctly, so if the following string read from the port corresponds to "*OK*". If everything works fine, the next command is sent, otherwise an error message is generated and registered on the Log file, and the software stops running.

Before returning, the function signals that the acquisition parameters setup has been successfully completed and saves the number of acquisitions per minute chosen (last parameter of the file, as shown in the previous paragraph) in a global variable that will be used in the function *acquisition.c* (following paragraph).

6.3.8 acquisition.c

This file contains the implementation of two key functions for the acquisition process:

1. *void* ***test_acquisition*** *(FILE * fp)*
2. *void* ***start_acquisition*** *(FILE * fp)*

The first function concerns the acquisition test to check if everything works fine with the radar, as already mentioned in paragraph 6.3.1: it takes as parameter the file pointer referred to the serial port and doesn't return anything. Essentially, the starting command is sent to the port that answers with "*OK*" followed by the measurement samples, if there aren't any problems, otherwise the software stops running and the error is reported within the Log file. During this phase the number of samples is actually saved in a global variable (for the 77 GHz radar it corresponds to 1001).
The second function contains the infinite-acquisition loop, taking again as input parameter the file pointer referred to the serial port and not returning anything. First of all a *double* type matrix of *acqPerMinute* (value saved during the serial communication setup, as explained in the previous paragraph) rows and *nSamples* columns is declared and it will contain the values returned from all the acquisitions performed within a minute. The time step between consecutive acquisition, within a minute, is then computed in this way:

```
step = round (60/acqPerMinute);
```

Where the *round* function simply returns the nearest integer of the argument inside the brackets. The real time R-values file is opened at this point and then the acquisition loop, repeated every minute, starts.
It is structured in such a way that every *step* seconds the following operations are performed:

- Trigger and start commands are sent to the port.
- If no errors came up, the samples of the current acquisition are saved in the corresponding row of the above mentioned matrix.
- The function waits until *step* seconds from the starting of the current acquisition have been passed and, if it is not a divisor of 60, when the last acquisition is performed, the function will wait until 60 seconds from the beginning of the minute have passed.

Then, after one minute:

- Z and R-values files are opened.
- The average between the *acqPerMinute* acquisition values is then computed per each of the samples previously memorized.
- The Z and R values are computed through a function I implemented within the *processing.c* source file and that I am going to present in the followinf paragraph.
- Z and R files for the current minute are closed.

These actions are then repeated from scratch.

6.3.9 processing.c

This file contains the implementation of three key functions for the processing, whose procedure has been explained in detail:

1. *void* ***load_parameters*** *()*
2. *double* * ***dft_computation*** *(double[], int)*
3. *void* ***ZR_computation*** *(double[])*

Since some mathematical operations are executed within this C source file, I had to include the Math library, developed for C language, in my Eclipse project. In particular:

- In C/C++ Build → Setting I had to include the library name (*m*) for the GCC linker.

Let's focus on the functions: the first one is essentially used to save the processing parameters read from the corresponding "*procPar.xml*" file, it doesn't take any parameters as input (thanks to the global variables usage) and it doesn't return anything. All the parameters mentioned in paragraph 6.3.6 are saved in an array, through the *libxml2* and *xmlManager.c* functions, and their values are then included in the global data struct I mentioned in paragraph 6.3.2. At this point, the number of FFT points to be used within the DFT, for the power spectrum computation, is calculated as the power of 2 closest to the number of samples, thus the radar performances are not degradated (as already explained in paragraph 5.3).
The next step concerns the sampling frequency computation, through the formula (27) illustrated in paragraph 5.3 and the frequency and range values vectors initialization.
Each element of the frequency vector can be computed using the following:

$$freqVector[k] = \frac{k\ Fs}{nFFT} \tag{31}$$

Then, each element of the range vector is computed in such a way (until the RANGE_MAX chosen and specified in *myglobal.h* is reached):

$$rangeVector[k] = \frac{freqVector[k]\ c\ T}{2\ BW} \tag{32}$$

The Range-values file is filled up at this point, with all the values outputted from equation (32). The RANGE_MAX has been set to 75 m, as shown with equation (29) in paragraph 5.3.

This first function of the *processing.c* library, is called within the *core.c* one, as already shown in paragraph 6.3.1.

The second function I implemented here, computes the DFT of an array of double values "*x*", given as input parameter, using *nFFT* points, specified as second input parameter, and returns its power spectrum, in dBm, as an other array of double values. First the array is filled up with 0-elements as padding, due to the fact that its size is lower than *nFFT*. Then, real and imaginary parts (*Xre* and *Xim* respectively) of each spectrum element are computed starting from the array elements with the followings:

$$Xre[k] = \sum_{n=o}^{nSamples-1} x[n] \cos \frac{n\ k\ PI2}{nFFT} \tag{33}$$

$$Xim[k] = -\sum_{n=o}^{nSamples-1} x[n] \sin \frac{n\ k\ PI2}{nFFT} \tag{34}$$

The procedure is obviously repeated for all the $k = nFFT$ elements of the DFT, that are also normalized through their division for the *nSamples* used. In such a way, it's possible to compute the power spectrum "*X* " of the array "*x*":

$$X = \sqrt{(Xre^2 + Xim^2)} \tag{35}$$

The actual power spectrum returned from this function is in dBm, therefore:

$$X|_{dBm} = 10log_{10}(X) + 30 \tag{36}$$

The third function of the library is instead used to compute Z and R values, and write them on the corresponding files: it takes as input parameter the double values array containing the averaged samples returned the different 1 minute acquisitions (infact this function is called within the last function of the *acquisition.c* library, presented in the previous paragraph) and report the Z and R values computes on their files.

The input array is first converted from digital units to power, as already shown in paragraph 5.3, to obtain: $Data_{Power} = \frac{(Data_{Volt}^2)}{10000}$. Then, the *dft_computation* function, described above,

is called using as parameter the power values array $Data_{Power}$ and its power spectrum is finally returned. However, since it is symmetric with respect to the origin (as property of the real signals spectrum), I took only half of the spectrum, corresponding to the positive frequencies.
At this point, Z, R and Realtime files are opened and all their values preceeding the minimum reliable distance are set to 0. Then, for what concerns the other Z values, the radar equation for metereological purposes is inverted to obtain them expressed in dBZ, while the R-values are extracted from equations (24) and (25), presented in the paragraph 3.5, with the addition of the radar calibration factor (if *flag77=1*), shown with equation (30). The R-values all also copied within the Realtime-values file, and before finishing, the function frees the memory allocated to the power spectrum returned in dBm, in order to have the actual last-minute values. Otherwise the same ones will always be reported, after every one minute loop.

6.3.10 myerror.h

This file contains the definition of all the possible errors that, I had thought, could occur during the software execution. I grouped them in various categories, corresponding each one to a library I implemented and I presented above. This file has been very useful, especially during the developing phase of the software, but it could be, in general, during its execution, since, each time something wrong occur, the corresponding error can be read on the Log File and the problem can be easily individuated and recovered, even thanks to the functionalities I implemented in *mydebug.h*.
Now I am reporting the five different errors categories:

1. ERROR_SERIAL.
2. ERROR_LOG.
3. ERROR_XML.
4. ERROR_ACQ.
5. ERROR_PROC.

In the following **Table 6**, I am instead going to focus on each possible error, within the different categories, describing it:

Table 6. Software Errors

Error ID	Error Description
ERROR_SERIAL 1	Unable to open the serial port.
ERROR_SERIAL 2	Unable to get attributes of the serial port.
ERROR_SERIAL 3	Unable to set INPUT BitRate of the serial port.
ERROR_SERIAL 4	Unable to set OUTPUT BitRate of the serial port.
ERROR_SERIAL 5	Unable to set attributes of the serial port.
ERROR_SERIAL 6	Unable to close the serial port.
ERROR_SERIAL 7	Unable to write on the serial port.
ERROR_SERIAL 8	Unable to read from the serial port.
ERROR_SERIAL 9	Unable to correctly SETUP the serial port.
ERROR_LOG 1	Unable to open the LOG file.
ERROR_LOG 2	Unable to write on the LOG file.
ERROR_LOG 3	Unable to close the LOG file.
ERROR_XML 1	Unable to correctly parse the XML Document.
ERROR_XML 2	Unable to correctly acquire the Root of the XML Document.
ERROR_XML 3	Unable to correctly read the content of the text node.
ERROR_ACQ 1	Unable to complete the Acquisition Test.
ERROR_ACQ 2	Unable to correctly set the TRIGGER
ERROR_ACQ 3	Unable to acquire the current value.
ERROR_PROC 1	Unable to initialize Range or Frequency vectors.
ERROR_PROC 2	Unable to initialize Power Spectra vector.
ERROR_PROC 3	Unable to initialize Z or R vectors.

This table concludes the overview of the software I implemented for the control system of the 77 GHz. All the functions have been presented and described in detail, and the different execution phases have been explained too.

The main advantage of this software resides in the fact that it is very versatile, as already mentioned in paragraph 6.3.7: it's enough to change the acquisition and processing parameters in the corresponding XML files and the software will output the correct power spectra values.

For what concerns the rain rate, instead, to obtain reliable measurements, the calibration of the radar currently used should be checked before, and then this should be inserted within the *processing.c* functions, as I did for the 77 GHz radar one. A little modification to the software will let it distinguish between different radar types, choosing the correct calibration for each of them.

Chapter 7

Data presentation: web page

7.1 Introduction and useful tools

Once I developed and tested (other test will be reported on the next Chapter 8) the software, I thought to present the data acquired from the radar in a simple, but smart way: I wanted to design a small web site, hosted by a local server installed on the mini pc, in which it could be possible to choose acquisition and processing parameters (modifying through a form *"acqPar.xml"* and *"procPar.xml"* files), and visualize data in real-time. The idea is that the radar should work in a fixed position for 24/24 hours, thanks to the software I developed on the mini-pc, and send back measurements which will be then processed and visible on the web-site, granting a real-time monitoring.

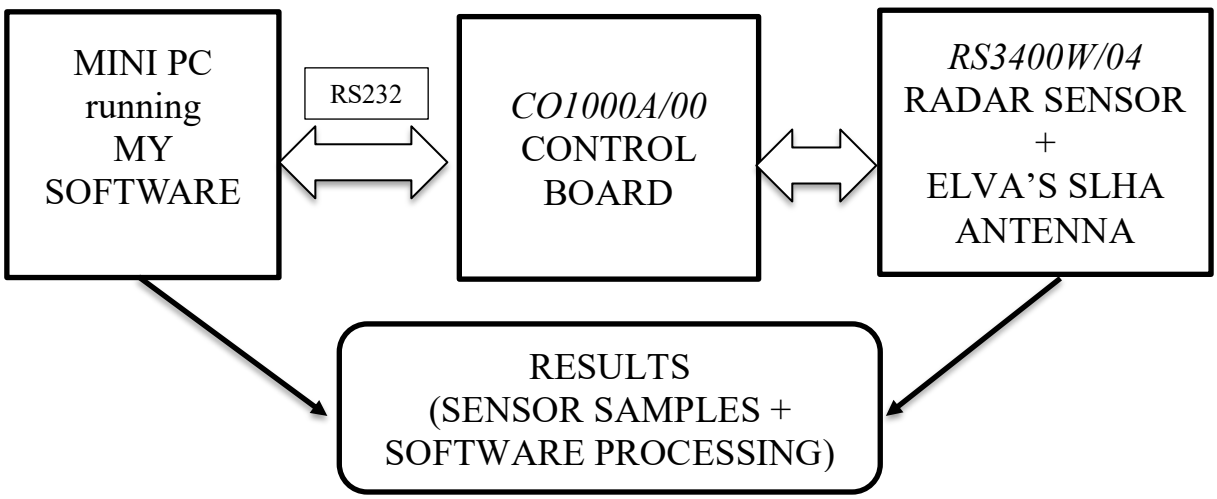
In order to realize this architecture, I need tools and programing languages, that I am going to briefly introduce:

1. *Apache HTTP Server*: one of the most famous and used open sourcer cross-platform server web. It essentially grants the installation of a web server on your pc able to host your personal web site and, on the following paragraph, I am going to explain in details how I configured it.
2. *HTML (HyperText Markup Language)*: standard markup language for creating web pages and applications. Any browser, acting as a client, receives HTML documents from a web server from a local storage and renders them into multimedia web pages. It is used as basis for the web pages and in particular, if it is the only language present within the document, it is able to provide just static content, thus the user can only see the web page received from the server, exactly how it is stored.
3. *CSS (Cascading Style Sheets)*: style sheet language used for describing the presentation of a document written in a markup language as HTML. It is designed to enable the separation of presentation and content, including layout, colours and fonts, in order to improve accessibility and flexibility. It also enables multiple web pages to share the same formatting.
4. *PHP (Hypertext Preprocessor)*: server-side scripting language designed for web development in order to realize dynamic contents. It can be embedded into HTML code, and it is executed by the web server itself that then sends the modified page to the browser and let the user see different content with resepct to the one originally stored in

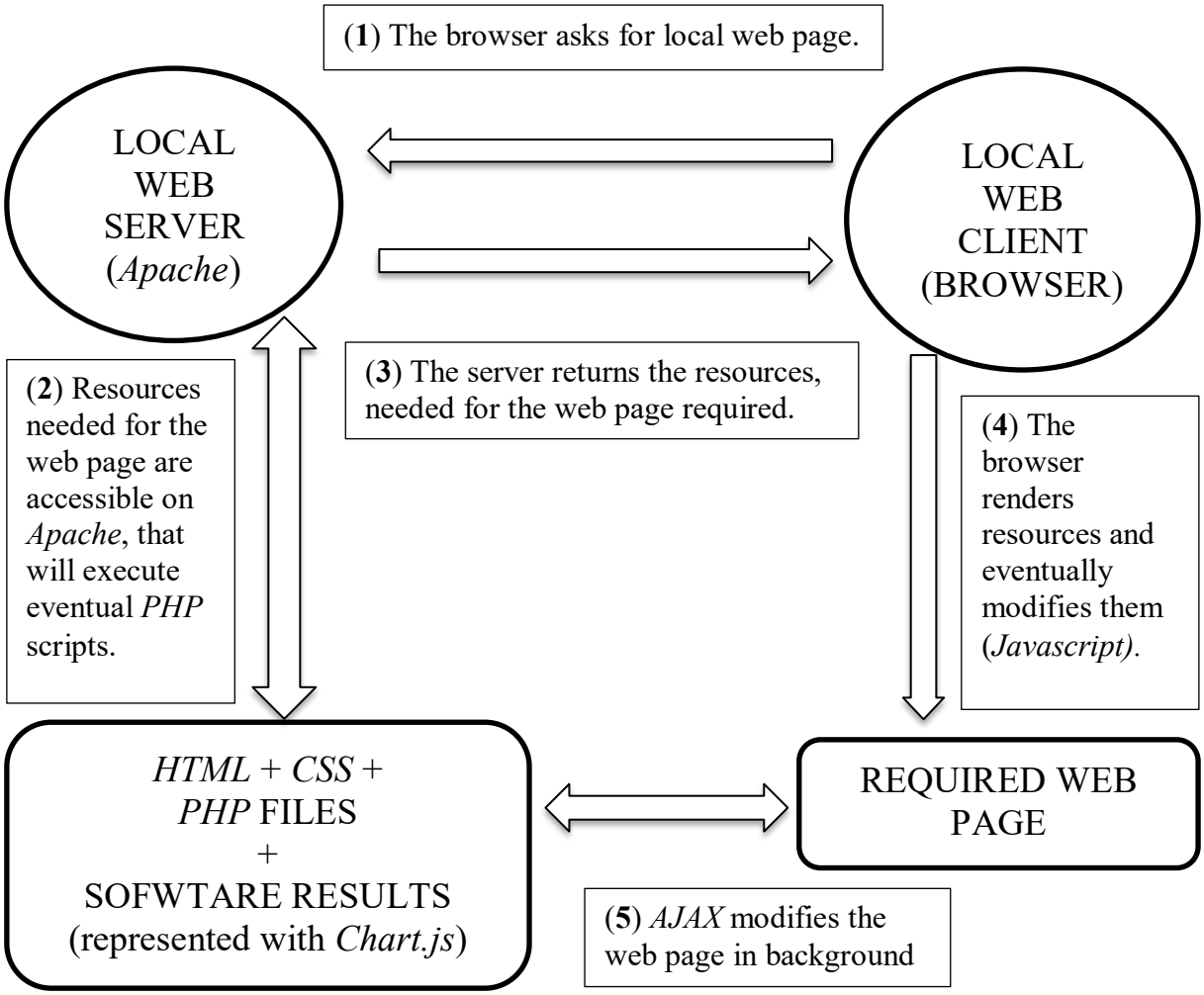
the server. Since a browser can just process HTML code, the server essentially modifies it, through PHP, to create customized web pages that are then sent to the user. In the following paragraph, I am going also to explain how I configured and I integrated it with the Apache Server.

5. *JavaScript*: mainly a client-side scripting language designed for web development in order to realize, as PHP, dynamic contents. With respect to PHP, it works directly on the web page that has already been loaded by the browser, to eventually change some interfaces behaviour, for example in response to mouse or keyboard actions or at specified timing. It can be incorporated within HTML code too and it grants more interaction with the user, since it works at the client side. I'll show how I had used it on my web pages, in the third paragraph of this chapter.
6. *AJAX (Asynchronous JavaScript and XML)*: client-side web development language used to create asynchronous applications, in such a way that server and browser exchange data in background (so asynchronously) without interfering with the display and the behaviour of the existing page. Therefore, the web page doesn't need to be refreshed to see changes on it. In order to use this language more efficiently, a popular JavaScript library, called *JQuery*, has been implemented and can be freely used. I'll show how I has used it on my web pages, on third paragraph of this chapter.
7. *Chart.js*: JavaScript library that can be used to create animated and interactive graphs to be included on web pages. It used HTML5 *canvas* element, essentially a container for graphics working with JavaScript. I needed for it, in order to create a graph to present the data acquired each minute. I'll show how I have used it on my web pages, again in third paragraph of this chapter.

The following block diagram summarizes the physical architecture I have realized:



The following block diagram summarizes the logical architecture for data presentation:



7.2 Web Server configuration

After giving an overview of all the tools I am going to use, in this paragraph, I am explaining how I configured the local web server on the mini-pc. In order to work not only at university, but even at home, I also repeated the same procedure on the Ubuntu OS hosted on my Windows PC by a virtual machine.

First of all, I installed Apache, through the following terminal command:

```
sudo apt-get install apache2
```

Apache2 refers to the second version of this software, published in 2000.

The main advantage on using Ubuntu is the fact that resources are easily accessible: infact in */etc/apache2/sites-available* folder, are located the configuration files for all the web sites hosted in the local web server. The default one is named *000-default.conf*.

To check if it is working, and if *Apache2* has been correctly installed, it's enough to launch the server through the following terminal command:

```
sudo service apache2 start
```

Then, on the search bar of the browser installed on Ubuntu (typically Mozilla Firefox), *localhost* must be digited (since it refers to the local web server) and the *Apache2* default configuration page, showin in the following **Figure 23** should appear.

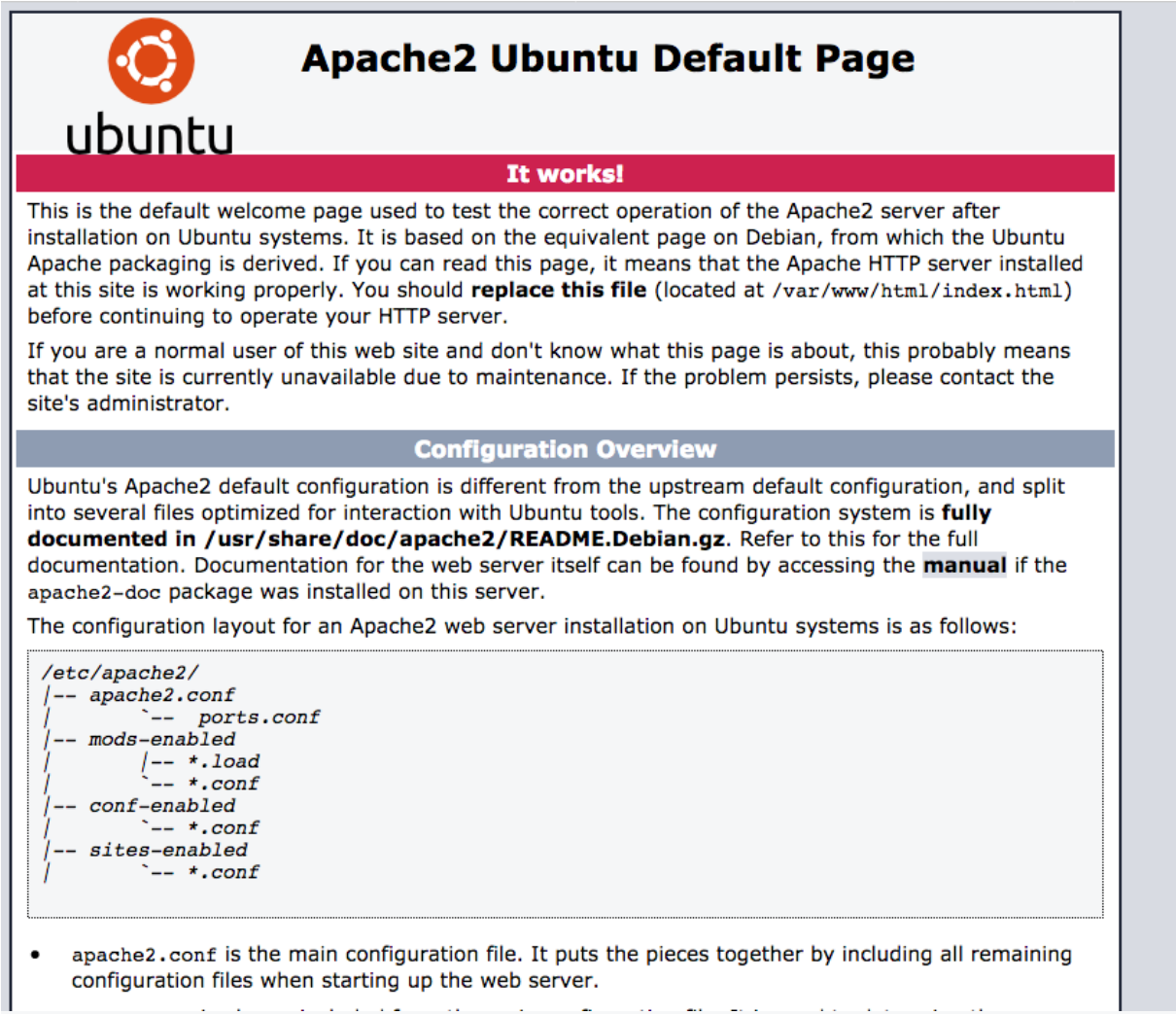


Figure 23. Apache2 Default Page

At this point, I can create my own customized web site, starting from the configuration file that should be copied from the default one and then changed according to needs. The terminal command to be used is the following:

```
sudo cp /etc/apache2/sites-available/000-default.conf
/etc/apache2/sites-available/mysite.conf
```

Now it's time to create the folder that will contain all the building files of my customized web site. Since the *Apache2* default ones are located in */var/www/html*, it's enough to create a new directory folder like this:

```
sudo mkdir /var/www/mysite
```

Then, the *DocumentRoot* defined in *mysite.conf* configuration file must be changed, since it must be linked to the new directory. The file can be opened with:

```
sudo gedit /etc/apache2/sites-available/mysite.conf
```

And after having applied the modification required, it should appear as shown in the following **Figure 24**:

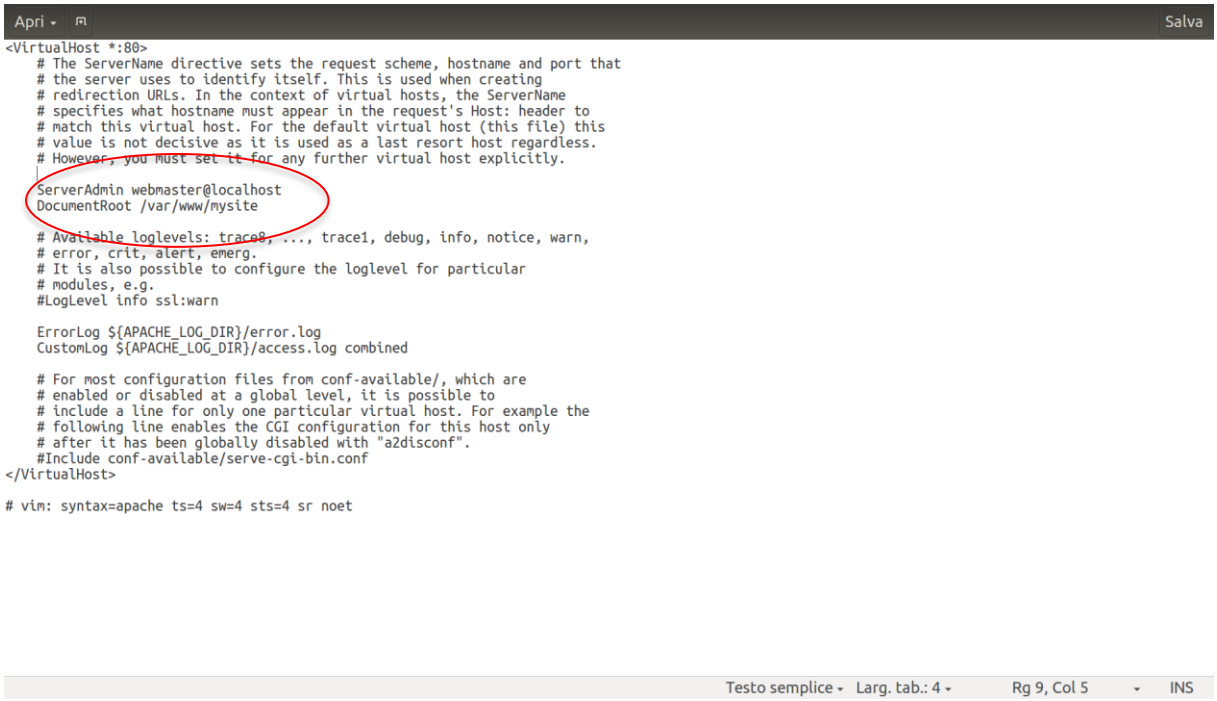


Figure 24. MySite Configuration File

Since I am working in particular directories owned by the system it self (such as */etc* and */var*), the super-user command *sudo* is always required. In addition, *Apache2* is owned by another user named *www-data*, so there is need to enable all users to the control of the configuration file and of the directory I created for my web site:

```
sudo chmod 777 /var/www/mysite /etc/apache2/sites-available/mysite.conf
```

In such a way there will not be any compatibility problems.

At this point, the default *Apache2* web site should be disabled and mine should be enabled. So, the following commands must be used:

```
sudo a2dissite 000-default.conf
sudo a2ensite mysite.conf
```

Each time modifications such these or modifications to the web site building files must be validated, this command must be digited:

```
sudo service apache2 reload
```

Before filling my web site folder with its building files, I need to correctly configure *PHP* for the *Apache2* web server. Therefore, I used this command:

```
sudo apt-get install php libapache2-mod-php php-mcrypt
```

Since the software input files are XML ones, I also need to install XML packages for *PHP*:

```
sudo apt-get install php7.0-xml
```

In addition, *Apache2* needs to be enabled to use the *libxml2* library and because of the different user owning it, permission must be changed:

```
sudo chmod 777 /usr/local/lib/libxml2
```

Finally, to avoid compatibility problems with *Apache2* web server, *PHP* safe mode should be disabled within the *php.ini* configuration file:

```
sudo chmod 777 /etc/php/7.0/apache2/php.ini
```

After opening it, I had to look for the *disable_functions* command and I disabled it, by commenting with an " # ", as shown in the following **Figure 25**:

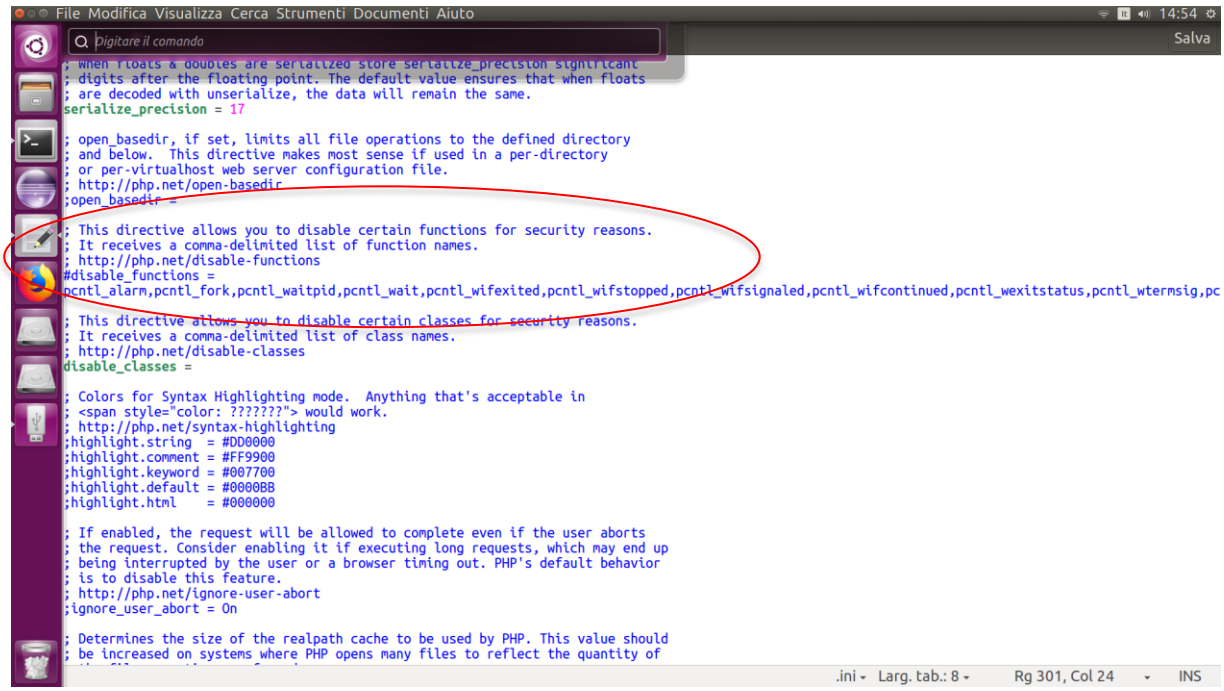


Figure 25. PHP.ini

Before starting with the web site development, I wanted to verify if it could be operative. So, in its folder (*/var/www/mysite*), I created the *info.php* file and I just put in the command *phpinfo();* returning information about the current *PHP* version installed. Then, I reloaded the server, as shown in the previous pages, and by digiting *localhost* on the browser I could see:

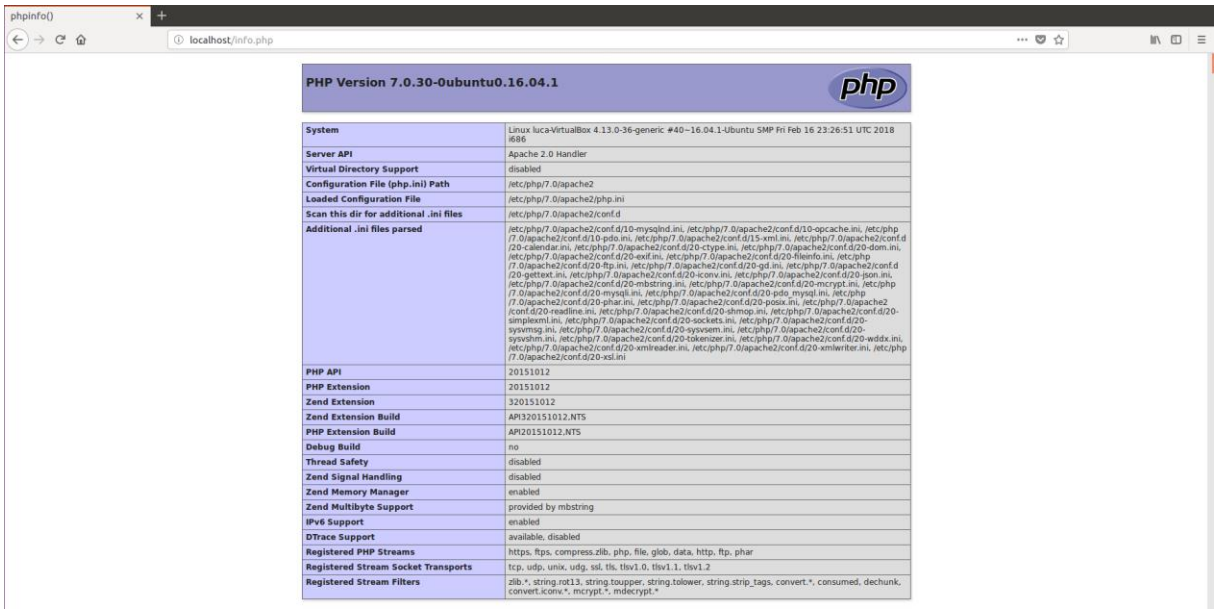


Figure 26. PHP Info

Everything works, so I'll show the web site development in the following paragraph.

7.3 Web Site development

First, I want to present the web site folder's organization. Then, I will explain in detail all the files I created.

I chose to place the files mainly containing *PHP* and *HTML* code, directly within the folder of the web site, thus */var/www/mysite*. The *CSS* file is located in the *styles* sub-folder. Finally, the *software* sub-folder contains:

- The executable file of the software I developed, since it can be started directly from the web site.
- The "*acqPar.xml* " and the "*procPar.xml* " files
- The sub-folder *results,* containing Z and R values files
- The sub-folder *log*, including the Log, Range and Real-time values files.

The following **Figure 27** and **Figure 28** summarize what I have just said:

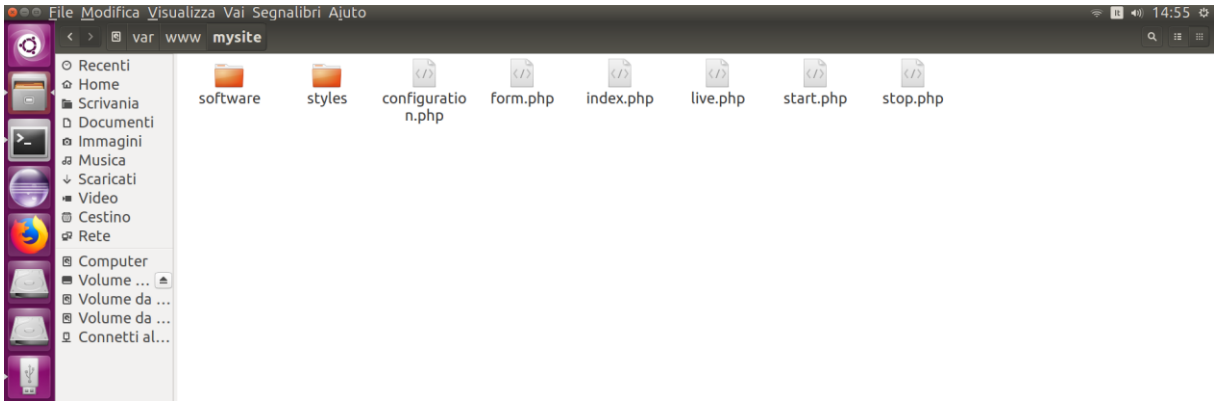


Figure 27. Web Site folder

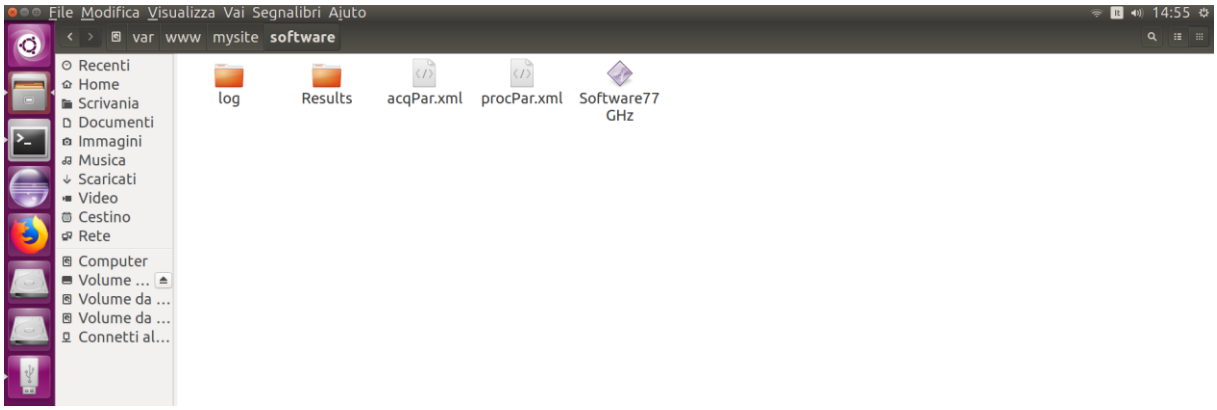


Figure 28. Software sub-folder

7.3.1 index.php

Going into detail with the three different web pages I designed, I am beginning with the *index.php* one. Since I had previously downloaded a free template, whose *CSS* file is located in *styles* subfolder, as mentioned above, I started working by modifying its layout and fonts.
The file is named *.php* even if it contains only *HTML* code: I simply wanted to use the same extension (the other pages all contain *PHP* code and need for *.php* to correctly work).
The code structure is quite basic:

- *<head>* section containing: the title tag too, the characters econding and the link to the *CSS* style sheet.
- *<body>* section including: the header with the link to the other two pages of the web site and the * * sections, where it's possible to read the instructions on how to configure and use the software, together with the web site.
- *<footer>* section including just copyright information.

The following **Figure 29** shows how it appears on the browser:

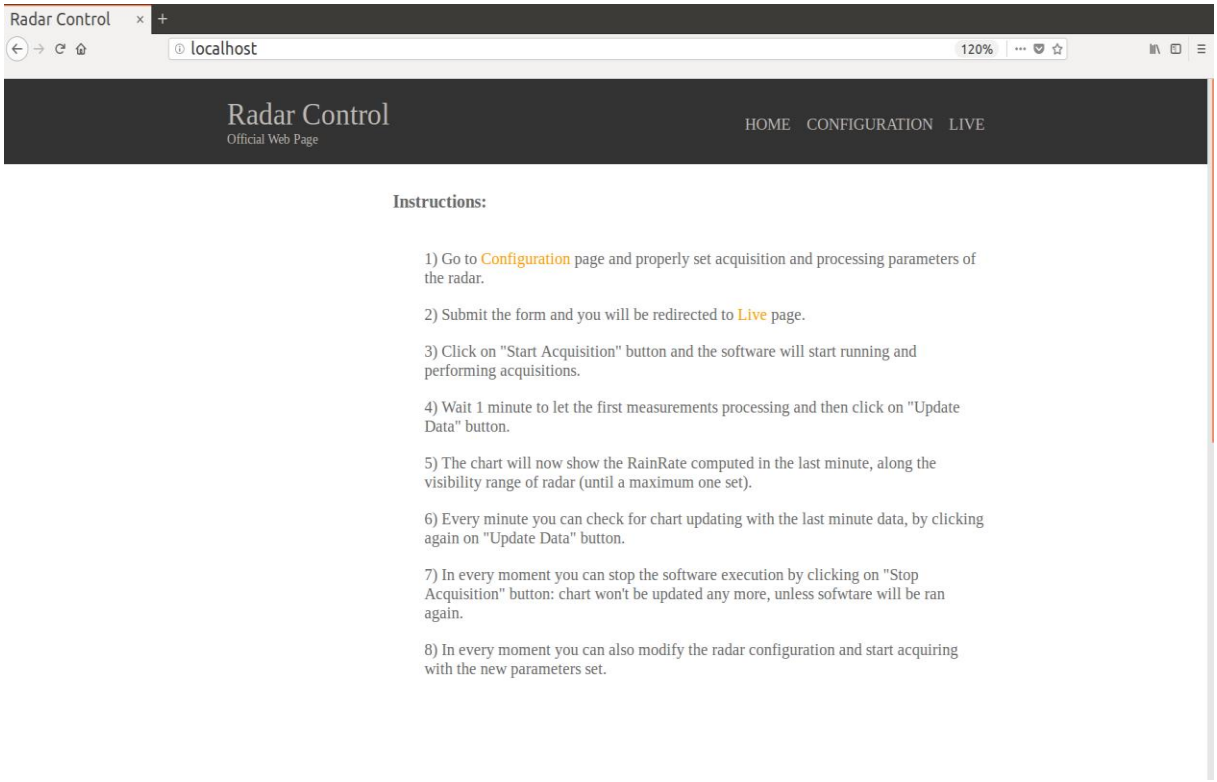


Figure 29. 'index.php' Page

7.3.2 configuration.php

Having read the instructions on the *index.php* page, a final user should go on the *configuration.php* page to compile the form proposed inside it, before running the software.
This page contains a code structure very similar to the *index.php* one but the main difference is represented by the <*body*> where the form is located. It let it modify almost all the parameters contained in the "*acqPar.xml* " and "*procPar.xml* " files, granting a totally personal configuration for the radar measurements.
The form uses the " *post* " method since it doesn't display the submitted data in the page address field (as instead the " *get* " method does), which is better for security reasons, and it has no size limitations. These data are then sent to the third web page I designed, the form-handler, named *live.php* and presented in the following paragraph, on my local web server.
The form fields can be filled up by writing directly values or by choosing between different options, according to what the field is referred to. However, if no options are present, an example of field compilation is provided through to the *placeholder* attribute, put within the <*input*> tag used to define a single form field.
In particular, for what concerns the acquisition parameters, the fields to be all mandatorily filled (thanks to the *required* attribute put within the <*input*> tag) are the following:

- *Radar Type*: its model must be specified with a string (Example: RS3400W).
- *Frequency Start*: I supposed two possible alternatives here, 9.5 GHz and 76 GHz since the controller board of the 10 GHz radar produced by SilverSima is the same (this shows how the software is versatile); it can be easily extended to other radars having the same board.
- *Frequency Stop*: coherently with what I just said above, I supposed two possible alternatives here, 10.5 GHz and 77 GHz; the final user should choose the stopping frequency, according to the starting one selected before.
- *Sweep Number*: I suggested 10 as the value, as already presented in paragraph 5.4.1; its range has been limited between 1 and 99.
- *Sweep Time*: I suggested 75 ms as the value, as also reported in the application notes of the radar sensor.
- *Acquisitions per Minute*: I suggested 6 as thevalue; its range has been limited between 1 and 60, to let the software at least have the time to process the data taken from an acquisition.

For what instead concerns the processing parameters, the fields to be all mandatorily filled are the following:

- *Radar Constant*: I suggested 69.15 as the value, whose computation has been discussed in paragraph 5.1.
- *Tx Power [dBm]*: I suggested 4 dBm as the value, since it is the typical one reported on the radar sensor technical specifications.
- *a*: coherently with that I derived on paragraph 5.2, I suggested 119 as the value.
- *b*: for the same considerations of the previous parameter, I suggested 0.67 as the value.
- *Minimum Reliable Distance*: I suggested 0.15 as the value, since it is the minimum possible, coinciding with the radar resolution.

Notice how, with respect to all the parameters defined in " *procPar.xml* " file, there are some of them that can't be modified by the form, and in particular they include: bandiwdth (<*BW*>), sweep time (<*SweepTime*>) and light speed (<*LightSpeed*>). The reason is that: the bandwidth is recovered as difference between *FrequencyStop* and *FrequencyStart* specidifed in the acquisition parameters above; the sweep time is taken from the same field specified in the acquisition parameters; the light speed is a fixed value.
All these data put on the form are sent to the *live.php* web page through the final *Submit* botton. The following **Figure 30** and **Figure 31** show how the *configuration.php* page appears on the web browser:

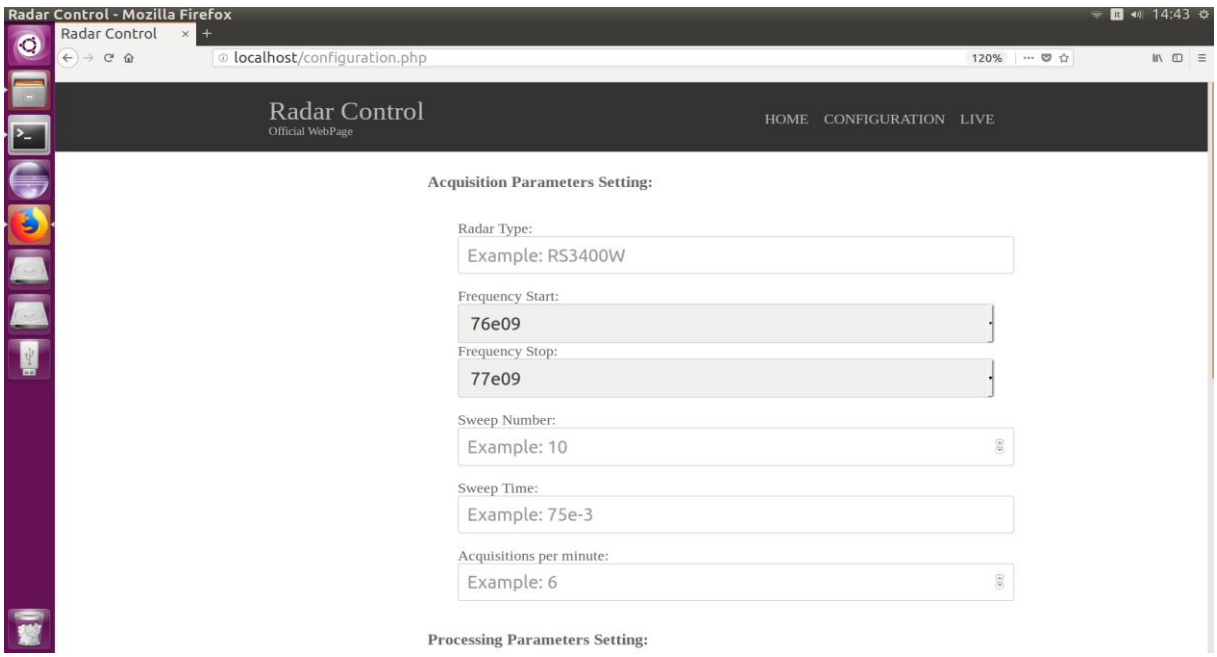


Figure 30. 'configuration.php' Page (Upper half)

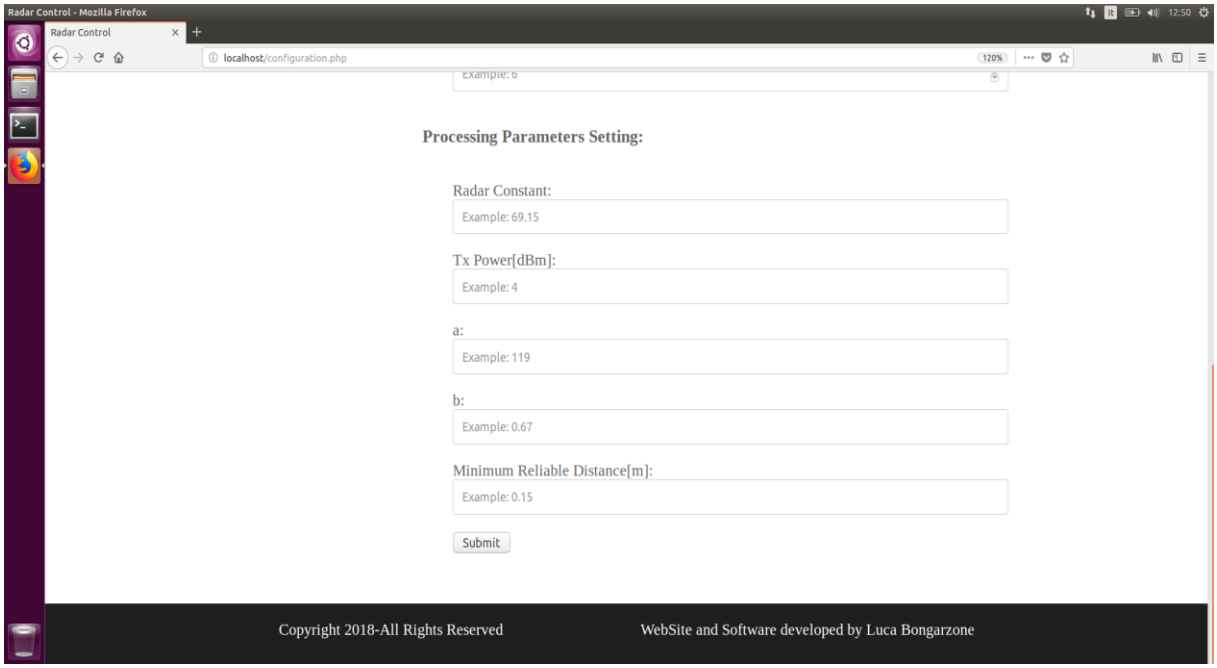


Figure 31. 'configuration.php' Page (Lower half)

In order to correctly process the data sent by the form, and substitute them in the right tags of the XML files, there is need for another *PHP* script named *form.php* and included in both *configuration.php* and *live.php* pages, respectively the form-container and the form-handler.
This script first verifies if acquisition and processing parameter XML files exist in the directories specified. Then, for each form field, checks if it has been submitted: every value sent corresponds to an element of the *$_POST* array (automatically generated by *PHP* once there is a request to the server web) thus it copied within the corresponding files.
In particular, for each text node of the XML file, the script executes the following operations:

1. Points to the next node path in the XML file.
2. Replaces its value with the one submitted in the respective form field, after the conversion to *HTML* special characters, through *htmlspecialchars* function.
3. Appends the value at the end of the file and closes the tag of the text node.

For what concerns the three values not modifiable from the form, I inserted it manually, through this script, in the right position, respecting the tag sequence defined in the XML files.
In addition, the script rewrites, at the beginning of each XML file, their first two lines specifying XML version, characters encoding and the root node (that is closed at the end of the file).

7.3.3 live.php

After having submitted the form, the web site directly takes the user to this third web page. Its code structure is similar to the other two pages, since the template is the same obviously, but there are some important differences:

- In the <*head*> there is the link to two external *JavaScript*, referring respectively to *Chart.js* and *JQuery* libraries, presented in paragraph 7.1.
- In the <*body*> is located the *JavaScript,* needed to create the table used to visualize the data outputted from the software, together with the *AJAX* code necessary to activate the functionalities of the three buttons of the page: *Start Acquisition*, *Stop Acquisition* and *Update Data* (actually invisible when *live.php* is downloaded for the first time by the browser). The table is inserted within the <*canvas*> *HTML* tag.

The following **Figure 32** shows how this page appears on the web browser whenever is visited for the first time and the software is not running, next I am going into details for what concerns the user interaction with the buttons:

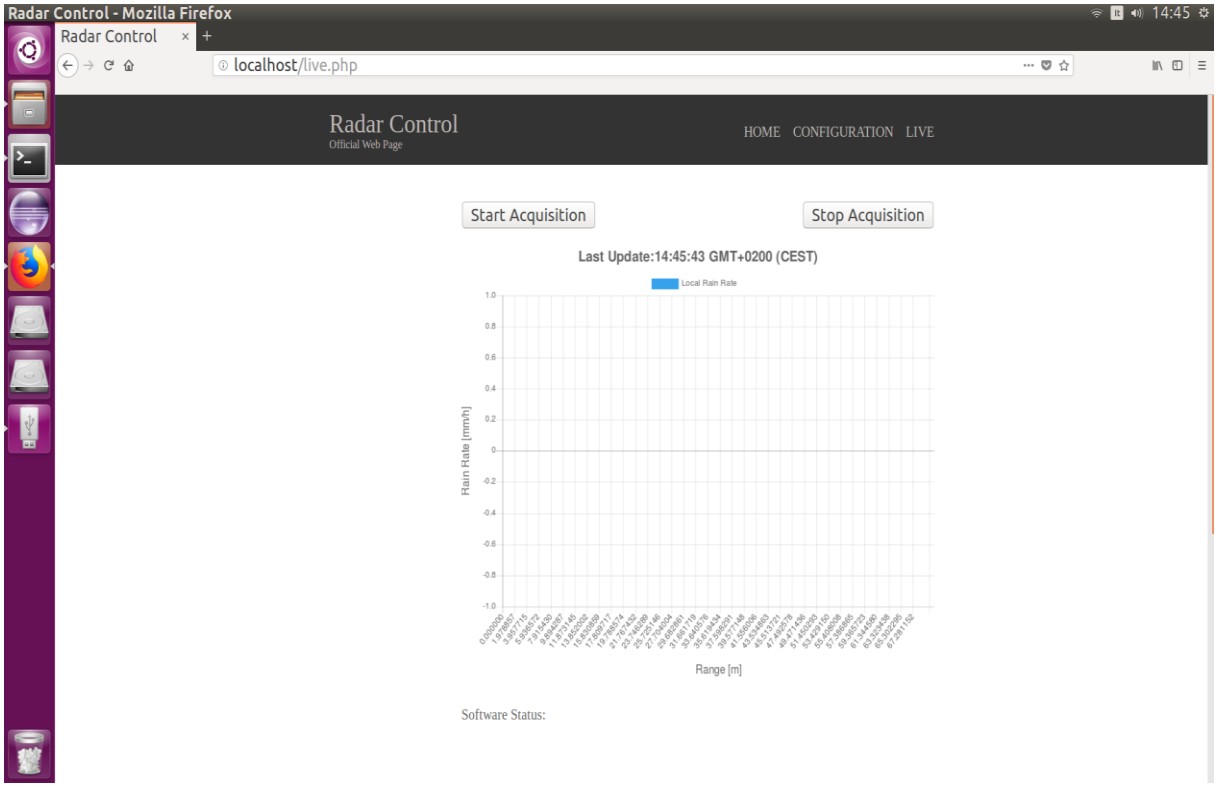


Figure 32. 'live.php' Page (No Data)

Whenever the page is loaded, thanks to the *window.onload* directive of *JavaScript*, the function located in the <*body*> script, executes its actions to create the table shown above.

First, the *AJAX* cache is disabled to avoid the browser showing always the same data previously downloaded and the *Update Data* button is hidden (infact in the **Figure 32** it is not visible).
Then, through the *get* function, the browser asks the server for two important resources: *rangeVect.txt* and *realTime.txt* files outputted by the software and presented in paragraph 6.2. Their values are indeed used respectively for the X-axis and the Y-axis of the table that is created once the document has been loaded.
If they are not available since the software hasn't been run yet, the table won't be shown. In the **Figure 32** the table has been created but it is empy and this means that just the *rangeVect,txt* file is present in the server. Layout and font options of the table can be modified through the script.
The time indication, located over the table, represents the instant in which the page has been refreshed or the table has been updated, and it can be used to point out the last minute data shown in the table referred to.
The following parts of the script describe the user interaction with the buttons. Once the *Start Acquisition* button is pressed, the script located in */mysite/start.php* file is executed. It simply runs the executable file of the software I developed, located within the sub-folder *software*, through the *exec* command. So, the software starts running in the background, as evidenced by the *"RUNNING"* word appeared close to "*Software Status"*, located below the table, and perform the first acquisitions. At the same time, the *Update Data* button is made visible:

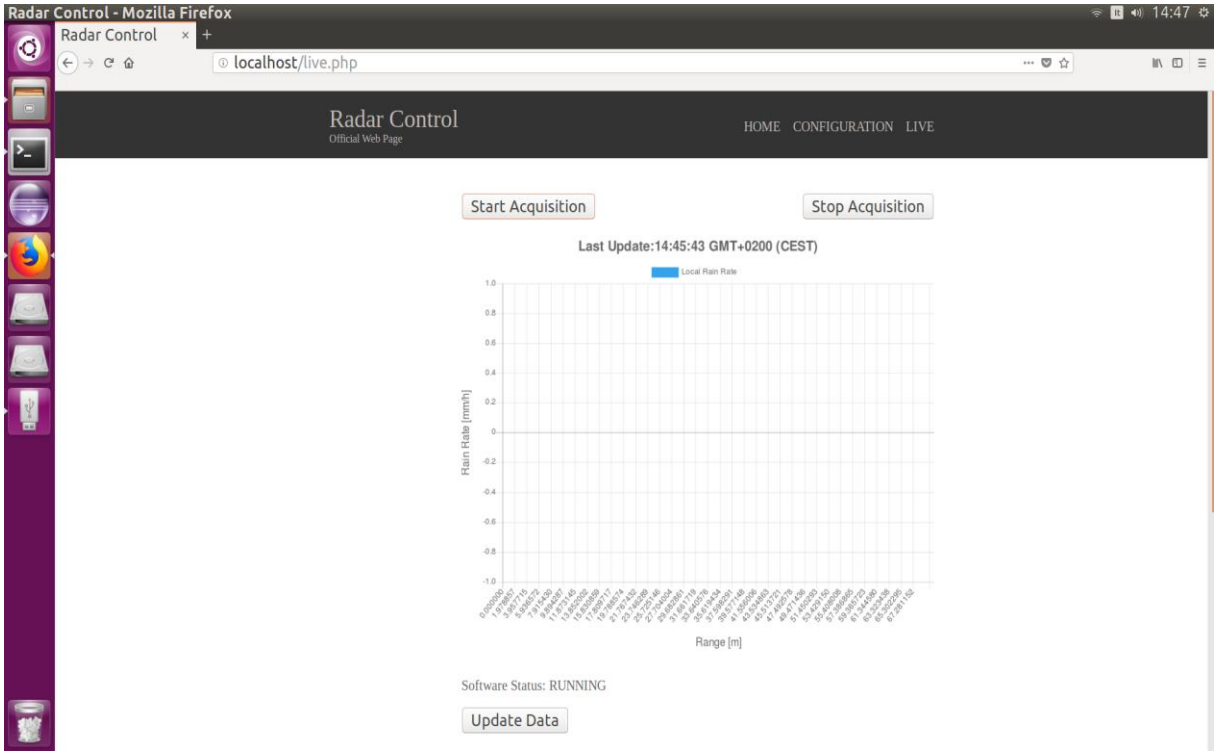


Figure 33. 'index.php' (Running)

Once, instead, the *Stop Acquisition* button is pressed, the script located in */mysite/stop.php* file is executed and the software stops running thanks to the *killall Software 77 GHz* command executed throught the *shell_exec* function of *PHP*. "*Software Status"*, changes in " *STOPPED*" and the button *Update Data* turns to be invisible.

Finally, once the *Update Data* button is pressed, the "*Software Status"* changes in *"RUNNING (Wait 1 minute before updating again)*" to remind the user that new data from the acquisitions performed by the software will be available after one minute. The *AJAX* cache is again disabled and the table is updated with the eventual new data taken from the two files located in the server (if available). Everything works without a real update of the web page, thanks to the *AJAX* asynchronous functionalities. The following **Figure 34** shows an example of filled table:

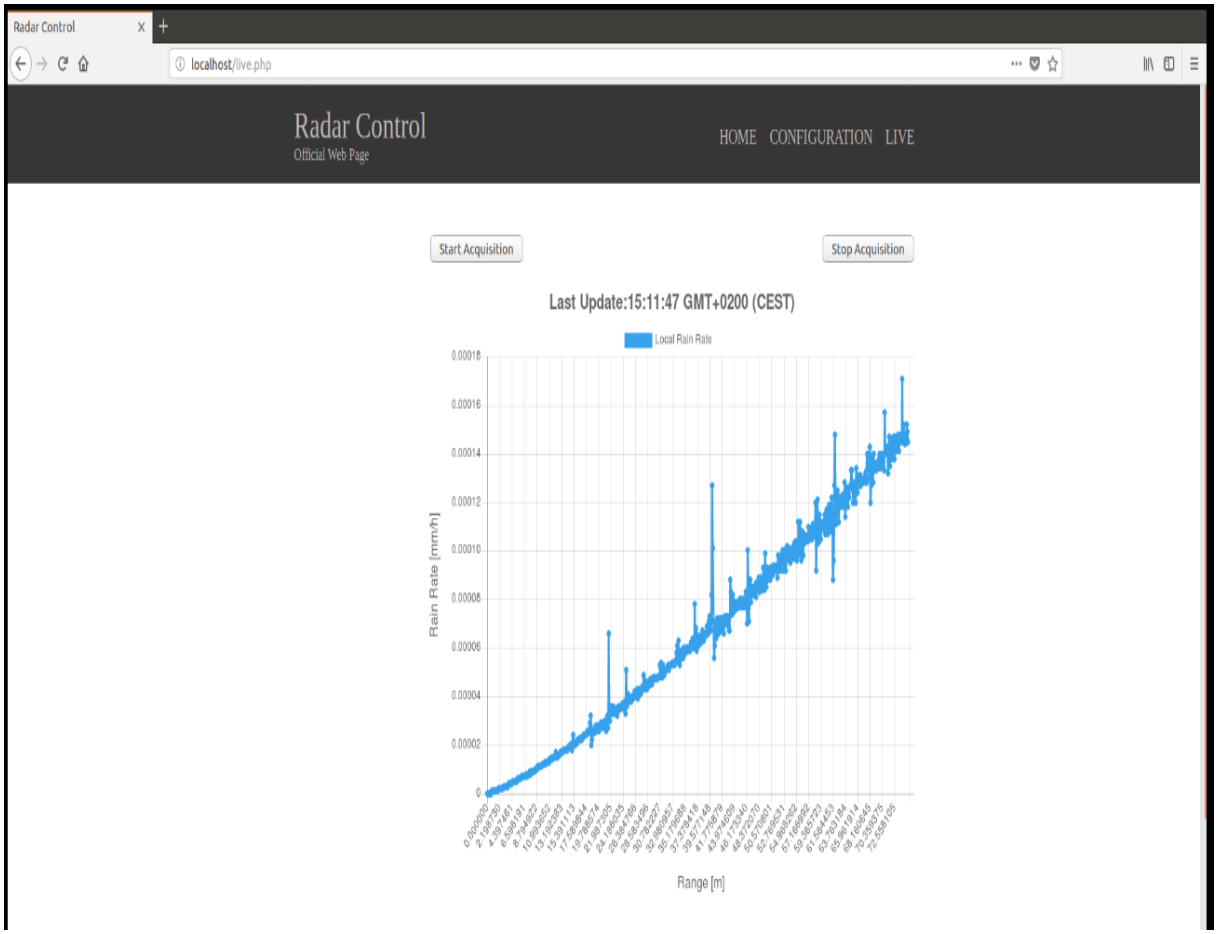


Figure 34. 'live.php' (Data)

These data refer to a not-rainy day as it can be seen by the practically null values of rain rates, along the Y-axis. The fact that the curve is quite ascending depends on the distance compensation ($20log_{10}(dist\,|_{Km})$) of the formula presented in paragraph 3.5, that is probably higher than the power received, in this case, since it wasn't raining and a few echos have been captured by thre radar sensor.

Now I am going to show the same *live.php* page after having substituted the data contained in *realTime.txt* with the rain rate values I computed through the Matlab code I wrote in November (after having obviously acquired data with the radar sensor), when I hadn't started the software and the web site development yet. Even if it was the same day of acquisitions, these values are not the same represented with Matlab in **Figure 22**, for the rain estimation test described in paragraph 5.4.3.

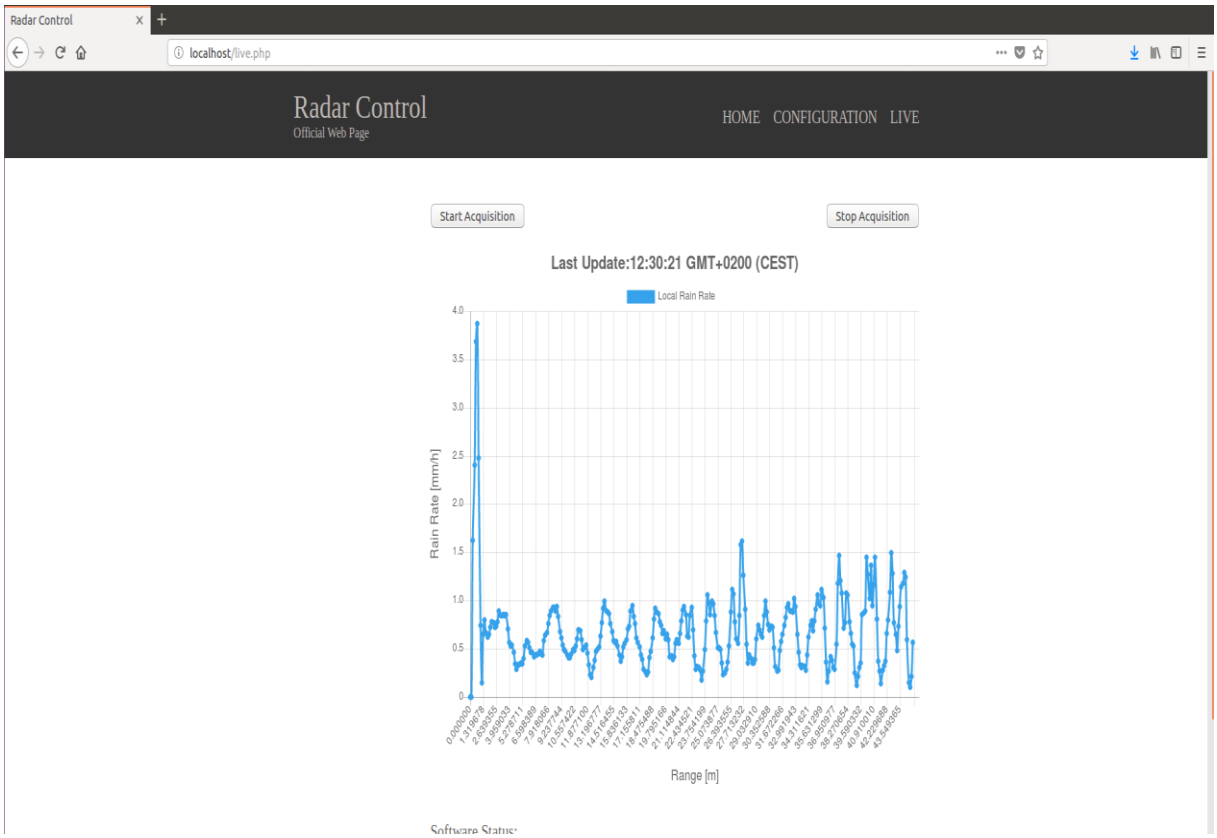


Figure 35. 'live.php' (November Data)

With respect to the previous image, the table here points out that, in the range included between 0 and 45 meters (this limitation has been already justified in paragraph 5.4.3 as also the initial peak clearly visibile), it was raining. The rain rate varies from about 0.1 and 1.5 mm/h. These values are clearly reasonable, considering a light rain as the one of that November day.

Figure 35 is another example of how the 77 GHz radar sensor could be used as rain gauge from the beginning of this thesis work.

Once I instead had developed the software and I had completed the web site, I could start acquiring data in real time. In the following Chapter 8, I will discuss in detail the main test I conducted after the preliminary ones.

I want to conclude this Chapter 7 with a last screenshot taken in real-time, during another day of acquisitions, of the *live.php* page, to show how the table has been correctly updated:

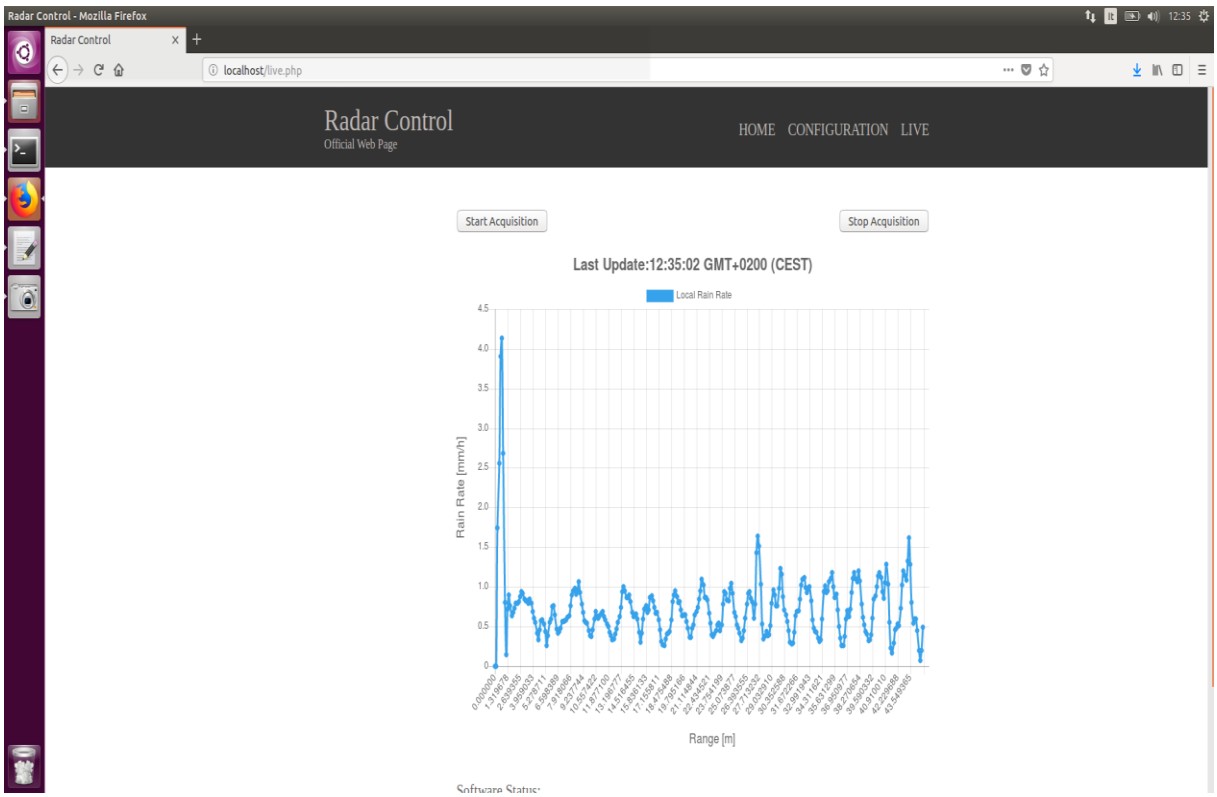


Figure 36. 'live.php' (Real-Time Data)

These data have been taken during a bit less rainy day than the one of November, presented in the previous **Figure 35**. However, the rain-rate values are still reasonable according to the precipitation type registered.

Chapter 8

Software test and measurements

8.1 Indoor Test

Once I had finished the software development, explained in detail within che Chapter 6, I started testing it as reported in the objectives of paragraph 1.1.
Test have been necessary to mainly understand:

- If the software could run without blocking.
- If the software was able to output everything I expected.
- If the data outputted were reasonable, thus if the computations I designed within the software were executed correctly.

At the beginning, as it usually happens in software development, there was always something not working. There have been problems especially in serial port communication, in the electronic board configuration and in the output file updating.
Luckily, I had designed the *debug.h* and the *myerror.h* files, that have let me easily find errors and problems, but it hasn'been always simple to solve them.
However, after some weeks of hard work I completed it and the first positive test I have led was inside the Politecnico's office where I developed the software.
First, I mounted the mini-pc over a support and I linked it to the radar sensor through the USB-Serial adapter. Then, I pointed the sensor toward the direction of interest and I started running the software through Eclipse platform, since I hadn't devolped the web page yet.
For this reason, the only way to visualize the outputted data resided on the Matlab code I had written for the preliminary test. So I chose to add two temporary output files from the software including the values of the samples returned during one acquisition of the radar sensor and the values of their corresponding power spectrum; in such a way I have been able to check, if they were reasonable in terms of correct obastcle detection, within the office, and if the power spectrum values returned by the software coincided with the ones derived from the Matlab code processing. For this test I decided to ignore on purpose the rain rate values.
The following **Figure 37** shows the measurement equipment setup, very similar to the one performed for the rain gauge test describe in paragraph 5.4.3:

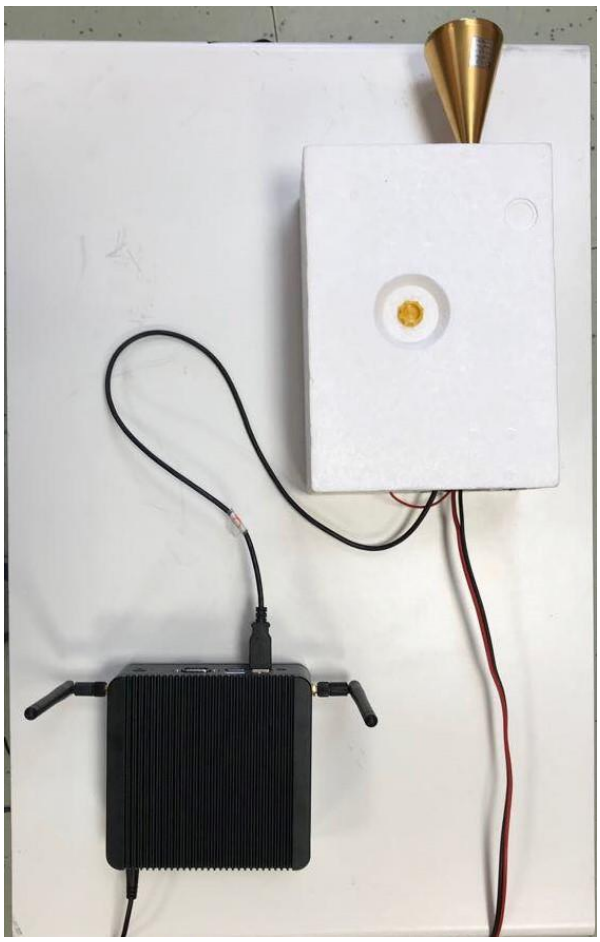


Figure 37. Software Indoor test: setup

The electronic control board and the radar sensor are located within the white box, from which it can be visible the antenna.

The following **Figure 38** shows instead the indoor environment toward which the antenna was pointed:

Figure 38. Software Indoor test: environment

I took the samples outputted from one of the acquisition files and I loaded it on the Matlab script I wrote. The first two plots it outputted are shown below:

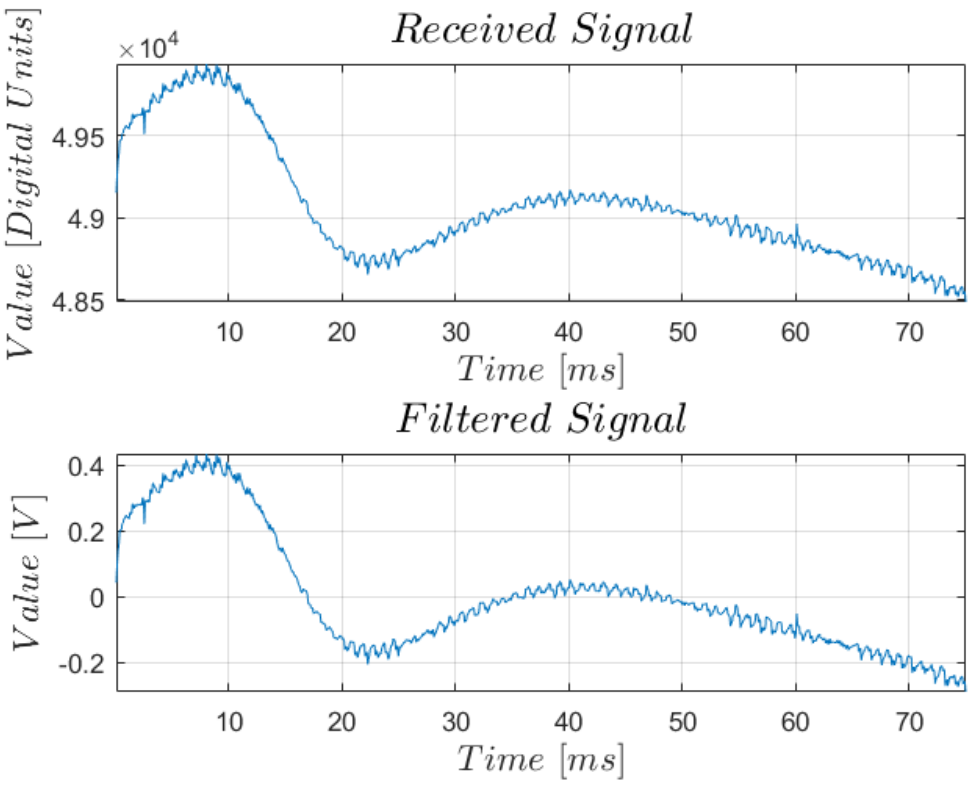


Figure 39. Software Indoor test: signal

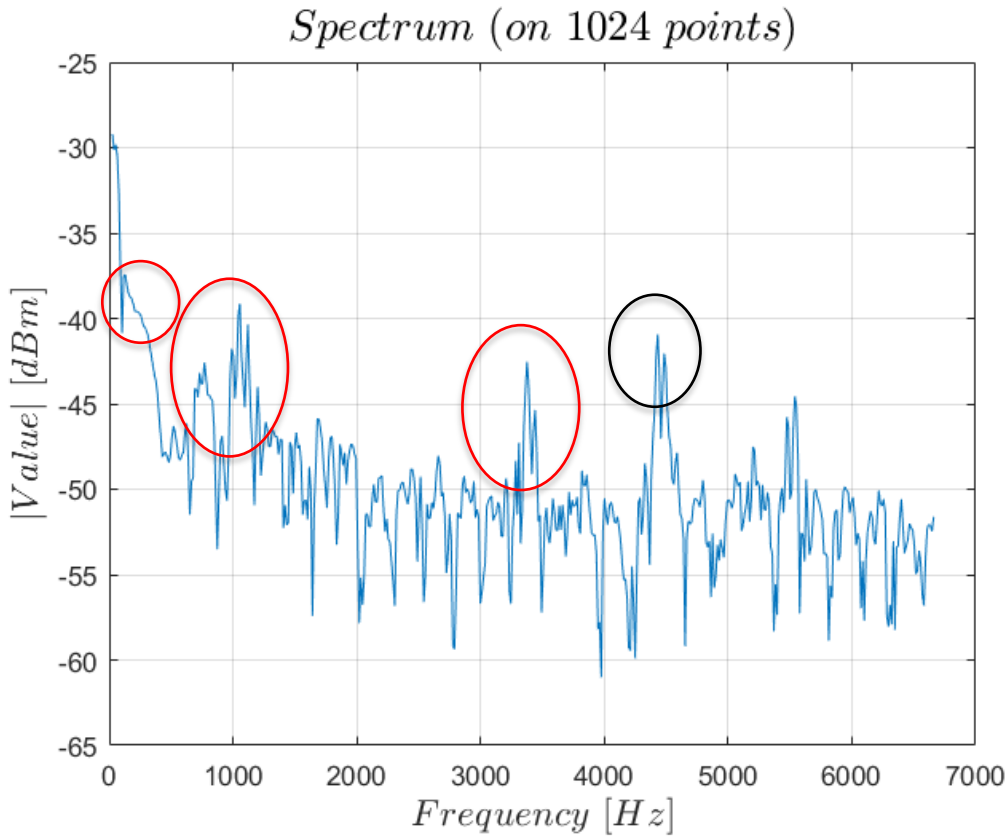


Figure 40. Software Indoor test: spectrum

The **Figure 39** shows, on the top subplot, the IF output signal, of the radar sensor, evolving within the sweep time, after its values have been averaged for the 10 sweeps (essentially it's the $Data$ I mentioned in paragraph 5.3). On the bottom subplot, instead the same signal, converted in Volt is represented (essentially $Data_{Volt}$ in paragraph 5.3) and it can be noticed how, as it happened in the preliminary test, its a 500 mVpp signal, as the specifics report (**Table 1**). These two signals are a bit noisy due to the large number of obstacles within the coverage zone of the radar.

The **Figure 40** simply gives the spectrum (just the positive frequencies, obviously, since it's symmetric) of the $Data_{Power}$ signal (obtained from $Data_{Volt}$ after the transformations discussed in paragraph 5.3). The red-circled peaks within the plot are reported below, with the corresponding distances computed with equation (4):

- 117.3 Hz → 1.32 m
- 769 Hz → 8.65 m
- 990 Hz → 11.14 m
- 1056 Hz → 11.86 m
- 1121 Hz → 12.61 m
- 3376 Hz → 37.98 m

The peaks corresponding to the first two range values are the most reasonable ones, since the room was about 9 meters large, and probably respectively refer to the desks and the wall. The others probably derive by reflections of the other multiple obastacles located within the office. The black-circled frequency peaks (at 4431.5 Hz and 4483.6 Hz), are instead, a constant presence of all the measurements, as already precised in paragraph 5.4.2.

For what concern the power spectrum values, they are quite reasonable too, since they oscillate between -45 dBm and -40 dBm. In addition, they practically coincide with the ones outputted by the software I developed.

So, if the preliminary test took me to confirm the validity of the measurements outputted by the radar sensor, both for obstacle detection and rain estimation, this test evidenced how the software is able to work autonomously and output the correct results.

Obviously, this concerns just the obstacle detection, as the outdoor test described in the following paragraph. For what regards the rain estimation, I preferred to show the test results together with the web site in paragraph 8.3: in such a way I will prove the validity of both software and web site, that I developed.

8.2 Outdoor Test

Since the indoor test couldn't have a reliability percentage too high, due to the large number of obstacles causing reflections, I decided to lead a software test also outdoor.
So, I went over the Politecnico's rooftop and I mounted the equipment (same considerations done in paragraph 5.4.2 hold):

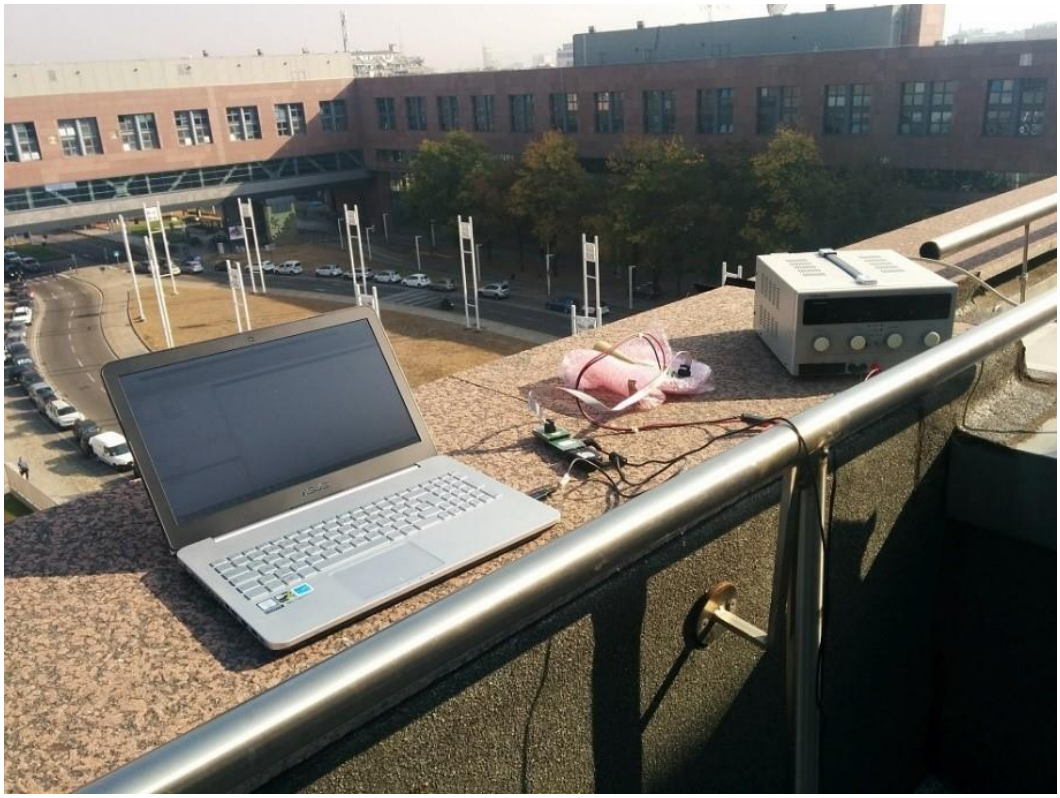


Figure 41. Software Outdoor test: setup

In the **Figure 41** above, it can be also seen on the right the radar sensor power supplier, able to provide the 12V needed.
I chose to point the radar sensor toward the ground, to check if it could be detected correctly. No other obstacles were present between them, so I expect the signal outputted to be very clean. I essentially repeated its test using my software and in the following **Figure 42** I represented the working environment and in particular the radar sensor direction of interest:

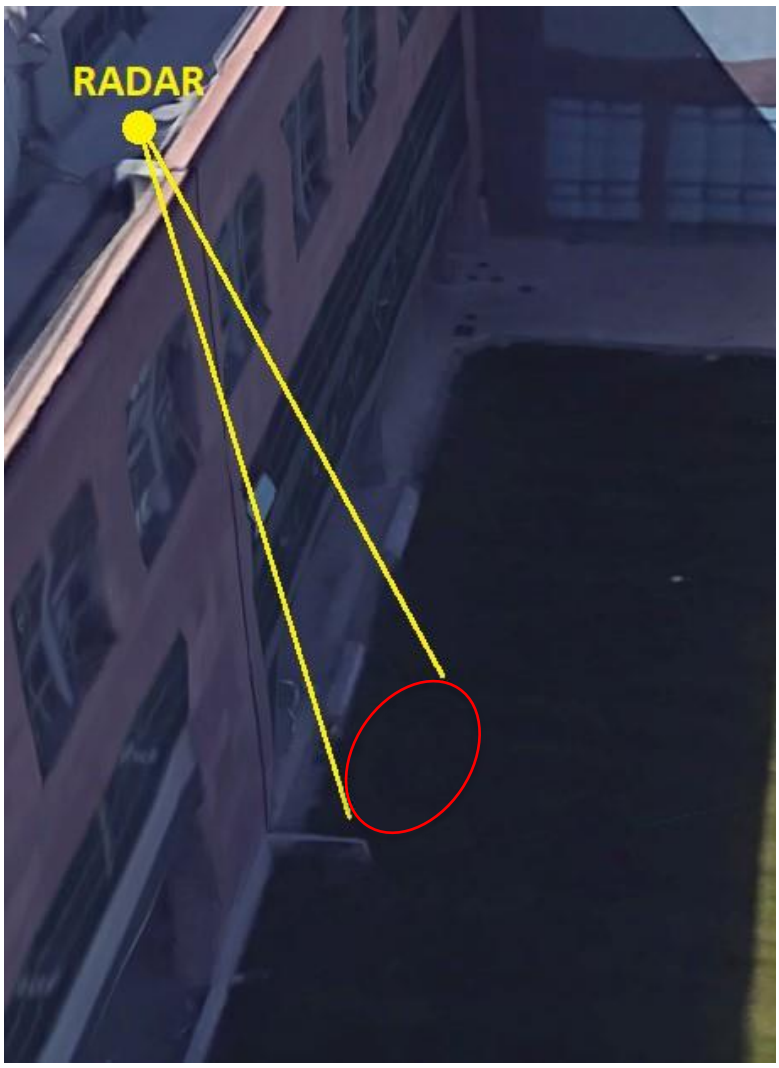


Figure 42. Software Outdoor test: environment

The red-circled area is essentially the one that should be individuated by the radar and, in order to check if the power values outputted and processed can be reasonable, I decided to previously compute a theoretical calculation about the RCS "σ_s" of the evidenced surface using the following formula (valid for a sphere in case of frequencies much higher than its radius "r", as in this case):

$$\sigma_s = \pi r^2 \tag{37}$$

In addition, the radr target in this case is a surface and not a volume, thus the formula (7) must be modified as follows:

$$P_r = \frac{P_t G_r G_t \lambda^2 \sigma_s}{(4\pi)^3 R^3} \tag{38}$$

The circular surface radius can be derived from geometrical considerations. Called "h" the perpendicular starting from the radar and considering half of the θ_{3dB}, the radius "r" can be also considered as the minor cathetus of the triangle whose hypotenuse is represented by one of the two yellow lines of **Figure 42**. Therefore:

$$r = h\ tan\frac{\theta_{3dB}}{2} \tag{39}$$

Since "h" corresponds to the range "R" (distance between the signal source and the obstacle) and it can be seen by Google Earth how it is about 27m, substituing all the values with the ones reported in **Table 3**, it results that:

- $r = 2.16\ m.$
- $\sigma_s = 14.65\ m^2.$
- $P_r = -51.36\ dBm.$

Now, I am going to show respectively the signal showing the samples values' evolution in time (as reported in the file outputted by my software) with its conversion mV and its power spectrum:

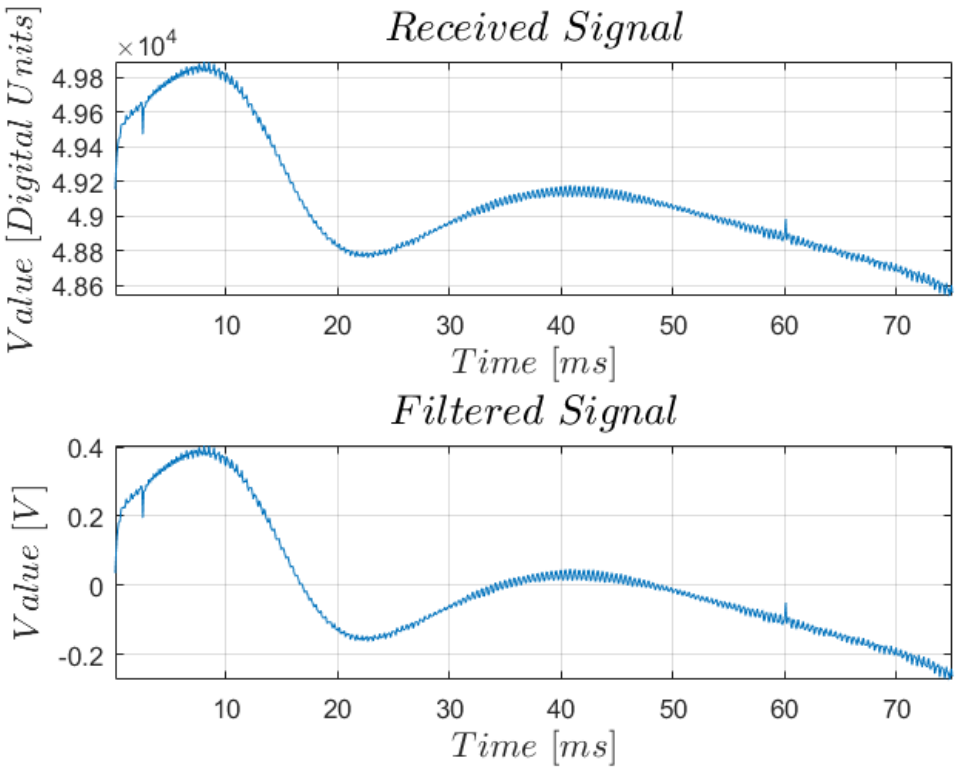


Figure 43. Software Outdoor test: signal

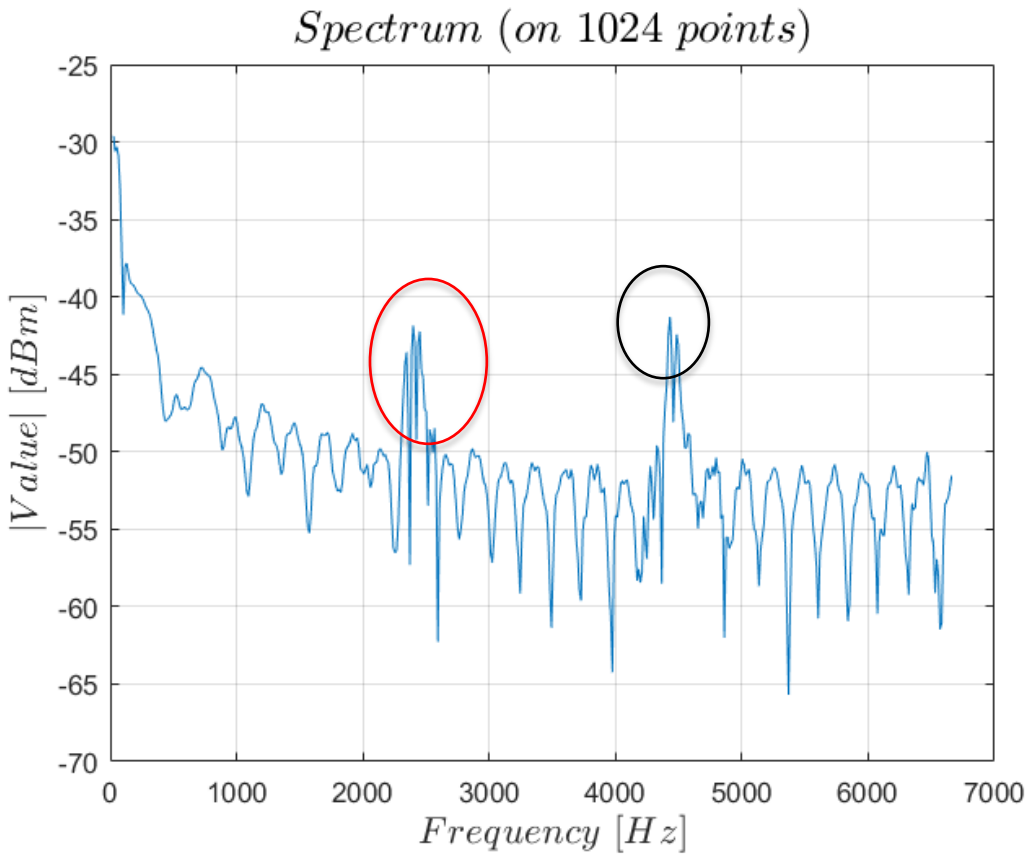


Figure 44. Software Outdoor test:spectrum

As can be seen by **Figure 44**, there are just two main peaks in the spectrum: the black one is the usual interference due to electronics, while the red one should represent the ground surface individuated by the radar. Converting the peak frequency of 2398 Hz, indeed, the distance returned is exactly equal to 27 m.

There is an important difference, however, on the power peak value: according to the dBm conversion of the signal returned by the radar sensor, is equal to about - 42 dBm. Therefore, the theoretical computations appear to be much more negative than the actual obtained results. Probably the reason for this discrepancy resides in a wrong consideration of the surface RCS, that it's probably higher. The theoretical gains considered, also, could be higher too, increasing so the theoretical received power. These are the only factors that could be changed, since the wavelength and the range are fixed.

However, since the target distance has been correctly individuated by all the test, I am sure the radar sensor is working as well as my software. The returned power values could be higher with respect to the theoretical ones, also for environment factors, difficult to compute.

8.3 Rain Test

At this point, I started testing the software for rain estimation's purpouse, as objective of the thesis. It was a rainy day of April, and I mounted the equipment, with my software and web site working too, in order to have the antenna pointing from the office window toward outside:

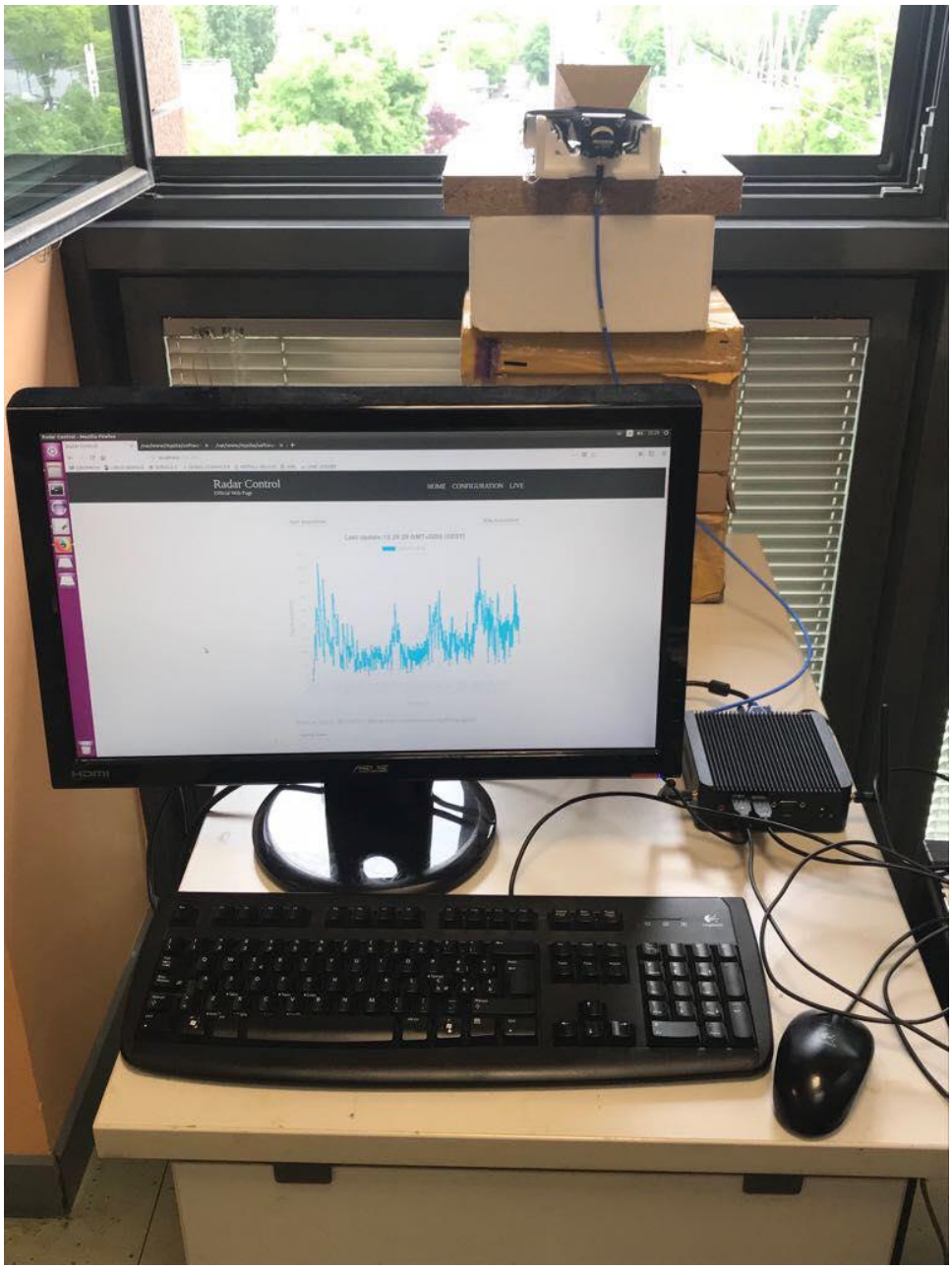


Figure 45. Software Rain test: setup

The image above is just to show how the radar sensor has been located. Actually, it represents the 10 GHz radar sensor and an attempt to calibrate it. The data represented on my web page are infact very noisy and not clear, meaning that it was an experimental test and nothing more.

In the test I am going to describe, obviously, the 10 GHz one has been substituted with the 77 GHz one.

The enviroment covered by the radar sensor is the same shown in **Figure 19**, in paragraph 5.4.3, since the rain test has been led from the same window. For this test, I again chose to output a file containing the samples returned from the radar, to be processed by my Matlab code, but I also chose to leave the software working completely, outputting also the files specified in paragraph 6.2. In such a way, I could compare the results, referring to a given acquisition, outputted by the software and represent on the web page, with the ones processed my Matlab.

Having already developed the web site, I started the software from the *live.php* page, as explained in paragraph 7.3.3, thus there was not need of Eclipse anymore. Then, I left the software working for more than one hour, thus it outputted different Z and R values files. By clicking on the button *UpdateData* I could experiment with changes on the table, as expected, since it represented the real-time data, referring to the last minute.

For the Matlab code, I chose one specific samples file to be processed:

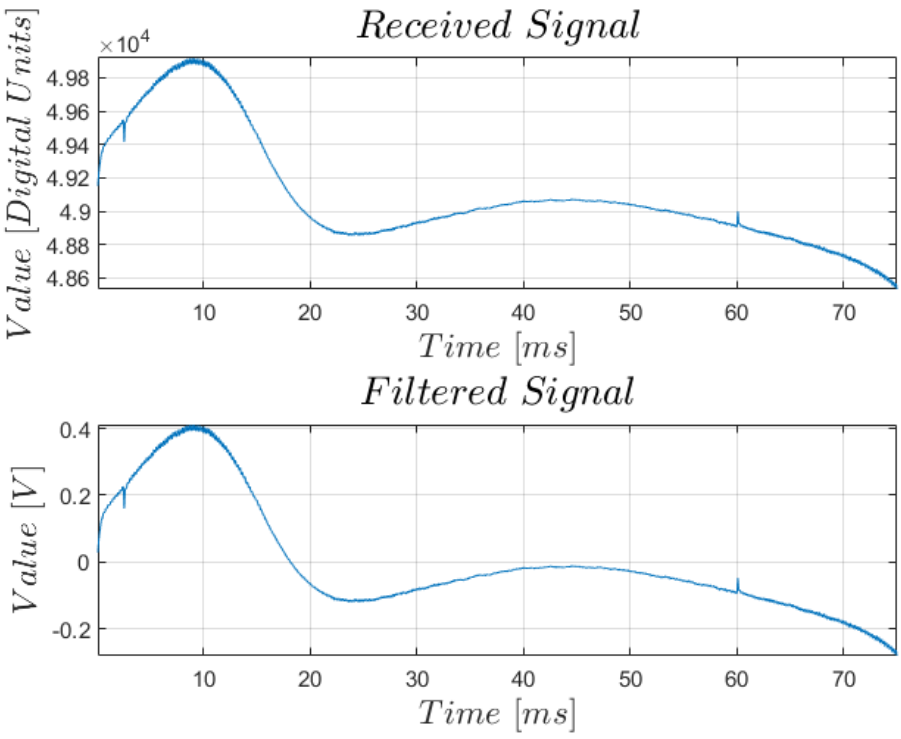


Figure 46. Software Rain test: signal

In the above **Figure 46**, I showed the signal derived from this file' samples. It can be seen how it is quite clean, with low noise and this can be explained by the fact that the main echo just came from rain drops, that are very small and so don't reflect too much power.

As I repeated in paragraph 7.3.3, the measurements range has been limited to 45 meters, thus the following plots referring to Z and R values will be limited to that value:

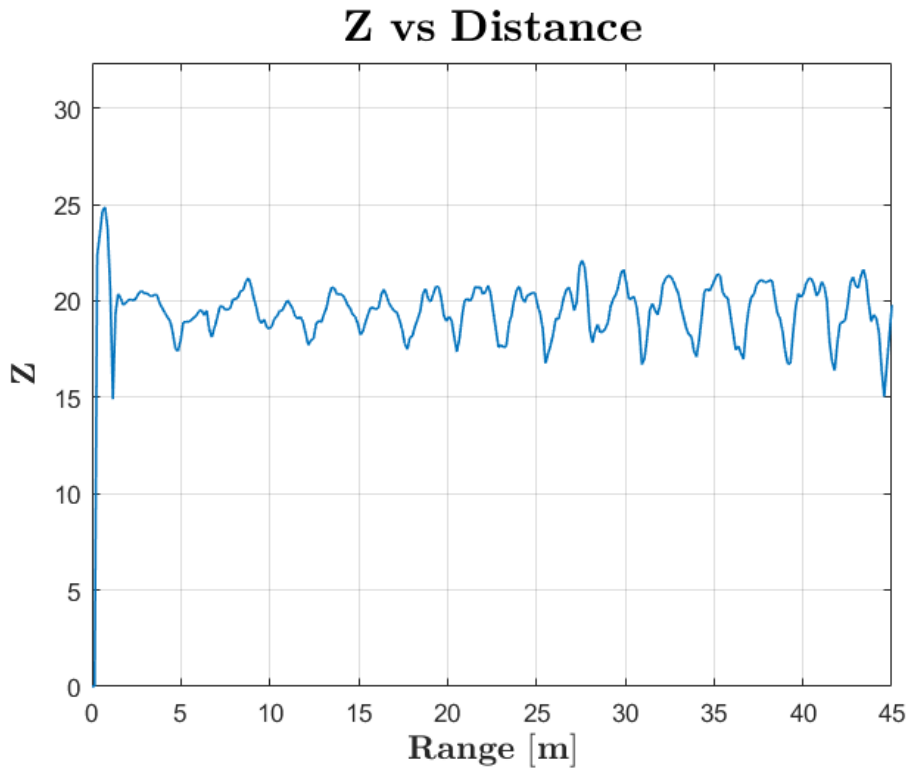


Figure 47. Software Rain test: Z values

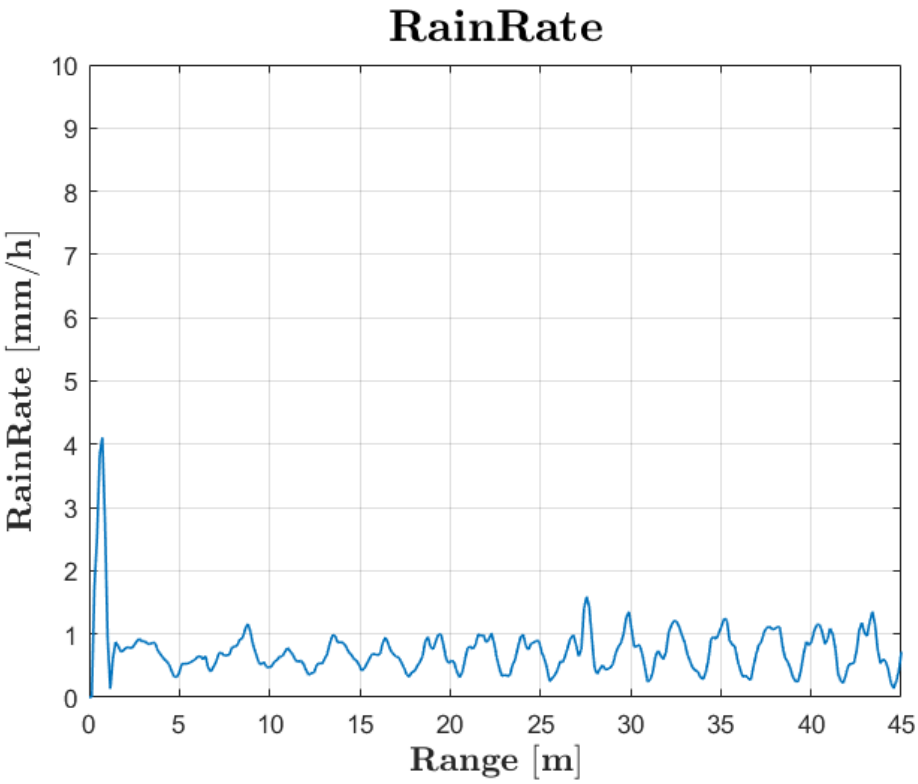


Figure 48. Software Rain test: rain rate values

Again, Z and R values are reasonable for the light rain that was falling. To check if the results processed by Matlab are the same visible on my web page, I am going to show a screenshot of it, taken during this considered acquisition:

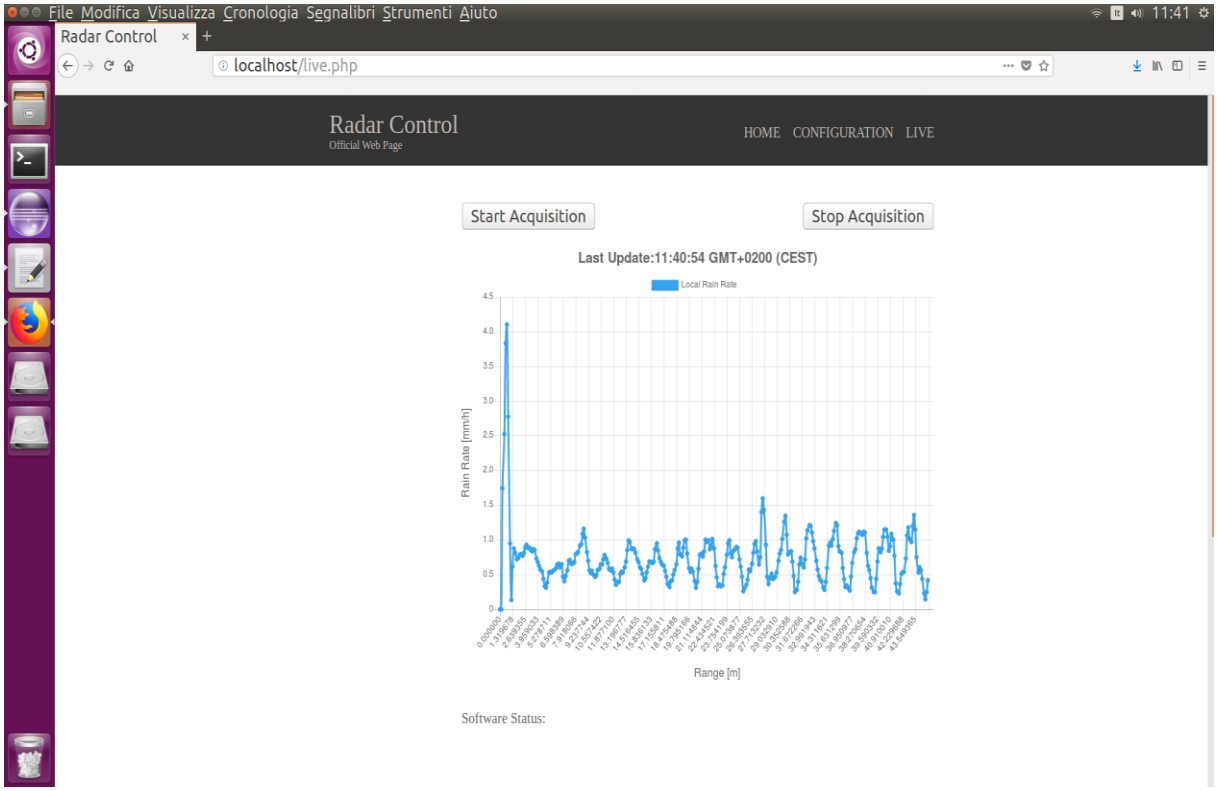


Figure 49. Software Rain test: 'live.php' web page

The image above represents a rain rate behaviour identical to the one of **Figure 48**. There are the same osciallations and the little peaks are located in the same points and have the same values (for example, close to 10 meters there is a little peak of about 1.3 mm/h and close to 25 meters there is another little peak of about 1.6 mm/h).

At this point, I can definitively conclude that my software outputs the correct results, since they are the same obtained with Matlab, concerning the rain estimation; in addition, these results are reasonable with the weather conditions registered during the acquisition phase and my web page works correctly in showing them in real-time while the software is running in background. These test have been a success and their results have been also presented to the EGU Conference 2018 which has been held in Vienna.

Actually, in order to show a cleaner graph than the one of **Figure 48**, I chose to average the rain rate values every 5 meters of range, supposing a constant value in each interval. In such a way, I obtained a stepped curve that can be seen in the following **Figure 50**:

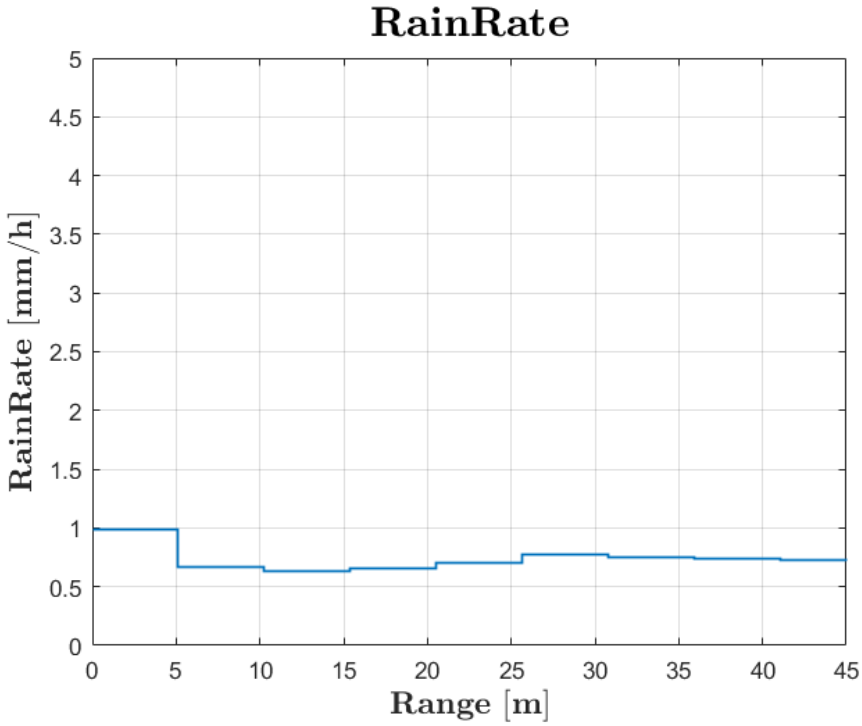


Figure 50. Software rain test: rain rate stepped

This type of curve, indeed, is much more immediate to understand and the rain rate is more visible within each range interval.

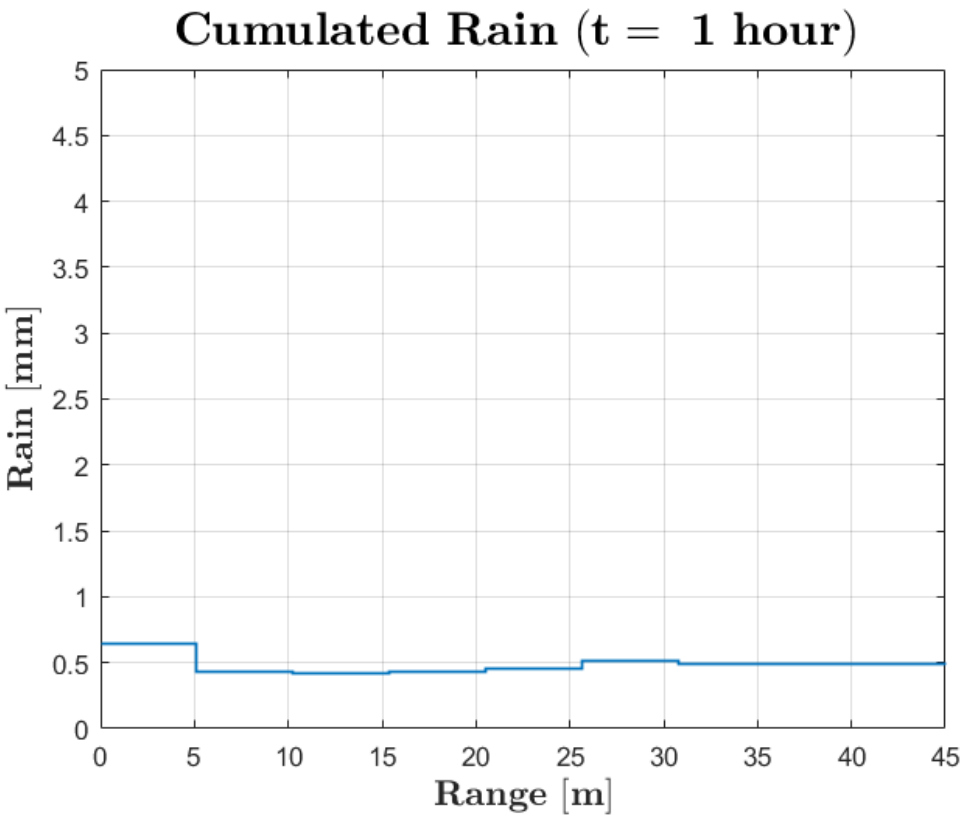


Figure 51. Software Rain test: cumulated rain stepped

The above **Figure 51** finally represents the sum of the measurements performed within 1 hour with a rate of about 6 acquisitions launched every minute (1 each 10 seconds; so all the measurements have been weighted for this time interval).
The rain accumulation during one hour results to be included between 0.5 and 1 mm and this is again reasonable, according to the weather conditions of that day.

8.4 Rain estimation preliminary test: results check

Since I couldn't use the radar sensor anymore, after the test presented in the previous paragraph, I would like to report here just a valid proof of the results obtained during the preliminary test. In particular, I am referring to the paragraph 5.4.3, where I described the radar sensor calibration and the first attempt to use it as rain gauge.
Even if the results shown in **Figure 22** seemed to be coherent with the weather conditions at the acquisition time instant, I verified on the Arpa Piemonte web site the rain amount for that day, in Turin, which was about 10 mm:

Valori aggregati giornalieri

Giornalieri | Mensili | Precipitazioni intense

Data	Precipitazione dalle 9 alle ...	Temperatura media (°C)	Precipitazione dalle 0 alle ...
03/11/2017	0,0	12,0	0,0
04/11/2017	0,0	13,0	0,0
05/11/2017	12,0	10,4	0,0
06/11/2017	9,8	8,8	21,8
07/11/2017	4,8	10,3	4,8
08/11/2017	2,6	9,1	0,0
09/11/2017	3,8	9,1	6,2

Figure 52. 6th November rain amount

Since the rain rate, which I found out through the measurements, was in average equal to 1.5 mm/h and since it has changed throughout the day (there were times when the rain rate was higher and times in which it was lower), also the preliminary test I led has been successful.

Chapter 9

Conclusions and future work

At this point, I can say that my thesis work has been successfully concluded.
Through these previous eight chapters I have shown how it has been possible to employ a 77 GHz radar sensor as a fully operative rain gauge (starting from the metereological theoretical models, the radar constant evaluation and the Z-R derivation, to finally lead the test). Also, I developed the control software, to let the rain gauge to work autonomously all the day, without blocking, and I developed a simple web page to show the results in real-time.
The three objectives presented in the first paragraph of this thesis have been achieved. The final prototype could easily be put in one of the white boxes, shown in the images representing the test setup, together with the mini pc containing my software and the local web site.
However, there is some future work that could be done:

- Checking of antenna gain and actual average transmitted power under different weather and environment conditions and after different consecutively working hours (these values, indeed, could change the randar constant, thus Z and R values outputted by the software couldn't be reliable anymore).
- Testing of the prototype within a car environment, to understand if it will eventually interfere with the other car sensors.
- Implementation of a web portal and/or a mobile application where it could be possible, for the final user, to visualize the real-time rain map, for a given area, realized with the data sent by different cars.

Bibliography

[1] A.Balanis C. "*Antenna Theory: Analysis and Design"*

[2] Allegretti M. *"Radar Metereology"*

[3] Belinda J.Lipa, Barrick Donald E. " *FMCW Signal Processing"*

[4] Berger Tor, Hamran Svein-Erik " *Gated FMCW SAR System"*

[5] Bertoldo S. *"Analisi preliminari radar meteo aeromobile"*

[6] Bertoldo S., Allegretti M., Lucianaz C. *"77 GHz automotive anti-collision radar used for meteorological purposes"*

[7] Bertoldo S., Allegretti M., Lucianaz C. *"Car as a moving meteorological integrated sensor"*

[8] Bertoldo S., Allegretti M., Lucianaz C., Perona G. *"Preliminary analysis to design a 77 GHz Wheather Radar"*

[9] Bertoldo S., Allegretti M., Lucianaz C., Perona G. " *Derivation of Z-R equation using Mie approach for a 77 GHz radar"*

[10] Bertoldo S., Allegretti M., Lucianaz C. *"Software defined radar"ù*[5] Brooker Graham " *Understanding millimetre wave FMCW radars"*

[11] Elva-1*"Standard Gain Lens Horns up to 220GHz"*

[12] ITU-R (Gas Attenuation) " *Recommendation P.676-11"*

[13] ITU-R (Rain Attenuation) " *Recommendation P.838-3"*

[14] Kemp Mattew James. "*A FM-CW microwave radar for rainfall applications"*

[15] Lockart. G.B. " *Digital Signal Processing for target detection in FMCW radar"*

[16] Matzler Christian*"MATLAB Functions for Mie Scattering and Absorption"*

[17] SilverSima*"FMCW Radar Sensors Application Notes"*

[18] SilverSima*"77 GHz Radar Sensor Data Sheet"*

[19] SilverSima*"Power and Controller Board Data Sheet"*

Ringraziamenti

Prima di tutto, vorrei ringraziare il mio relatore, Silvano, per il pieno supporto tecnico, e non, che mi ha dato nel corso di questi mesi di preparazione e svolgimento della tesi.
Inoltre, vorrei ringraziare i miei genitori, Elvira e Luciano, per avermi permesso di studiare lontano da casa, in questa prestigiosa Università, il Politecnico di Torino, e la mia fidanzata Elisabetta, per tutto il supporto garantitomi in questi due anni di studi magistrali.
Per finire, i miei amici più stretti e cari che da sempre fanno parte della mia crescita personale: grazie anche a loro sono riuscito a portare a termine questo percorso.