**Antenas 5G para acceso fijo a Internet con mejora de capacidad - 5G AFIANCE**
**THD 2019**



# 5GAFIANCE – Antenas 5G para acceso fijo a Internet con mejora de capacidad

## *5G antennas for fixed Internet access with capacity enhancement*

# E8 – IMPLEMENTACIÓN Y EVALUACIÓN DE ALGORITMOS DE CONTROL DE HAZ

## *D8 – Implementation and Evaluation of Beam Control Algorithms*

| | |
|---|---|
| Original Delivery Date: | December 31st 2020 |
| Actual Delivery Date: | March 8th 2021 |
| Editor: | Nokia |
| Work Package: | PT3 |
| Version: | 0.1 |
| Number of Pages: | 24 |

| Dissemination Level | | |
|---|---|---|
| PU | Public | |
| PP | Restricted to other participants of the program | |
| RE | Restricted to a specific group defined by the consortium | |
| CO | Confidential, only for the members of the consortium | X |

# E8 – IMPLEMENTACIÓN Y EVALUACIÓN DE ALGORITMOS DE CONTROL DE HAZ

| VERSION DETAILS | |
|---|---|
| Version: | 0.1 |
| Date: | March 8th 2021 |
| State: | Draft Version |

| PARTICIPANTS | |
|---|---|
| **Participant** | **Name** |
| Nokia | Aarón Garrido Martín |
| Nokia | Alfonso Fernández Durán |
| | |
| | |
| | |
| | |
| | |
| | |

| DOCUMENT HISTORY | | | |
|---|---|---|---|
| **Version** | **Date** | **Responsible** | **Description** |
| 0.1 | 08-03-2021 | Nokia | First Draft Version |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

| DELIVERABLE REVIEWS | | | |
|---|---|---|---|
| **Version** | **Date** | **Reviewed by** | **Conclusion*** |
| | | | |
| | | | |
| | | | |

*p.e. Accepted, Develop, Modify, Re-edit, Update

# Contents

# Figures

# Executive Summary

This document describes the beam control algorithms designed and implemented to carry out the characterization of the radio channel at millimeter waves. The implemented algorithms allow controlling the beam using a mechanical pointing system. This implementation is basic to automate the radio-frequency channel characterization process, and it also serves as the bases of the beam pointing control algorithm.

# Resumen Ejecutivo

Este documento describe los algoritmos de control de haz diseñados e implementados para realizar la caracterización del canal de radio en ondas milimétricas. Los algoritmos implementados permiten controlar el haz mediante un sistema de apuntamiento mecánico. Esta implementación es básica para automatizar el proceso de caracterización de canales de radiofrecuencia, y también sirve como base del algoritmo de control de apuntamiento del haz.

# 1 Introduction

Communication systems that transmit in the millimeter wave (mmWave or mmW) frequency bands use highly directional antennas so proper orientation of the antennas is a key factor for reaching the best performance of the system; a misalignment would drastically decrease the link capacity.

5G-AFIANCE project is focused on the user equipment (UE) side and is specifically focused on fixed wireless access (FWA) devices, that is, devices that will be statically placed at the users' premises. Two types of FWA deployments are considered, outdoors and indoors.

Different antenna system solutions for the UE are proposed depending on the type of FWA deployment, and each antenna solution uses a different pointing system mechanism. The pointing mechanisms is also different depending on the frequency band:

- For indoor deployments, 5GAFIANCE project proposes to use a mechanical pointing system for coarse pointing and an electrical pointing system for fine pointing. The coarse pointing direction is calculated to maximize the performance at mmW frequencies. Once the mechanical system is oriented, a fine pointing tuning can be obtained using an electrical pointing system (electrical beamforming). In this case, the mechanical system automatically finds the best direction.

- For outdoor environments, a combination of mechanical and electrical pointing system is also proposed, but in this case, the mechanical pointing system is not automatically set by the antenna system. The coarse pointing must be known before installing the antenna system. Once the customer or the installer know the coarse pointing direction, the antenna can be placed, and the fine tuning is done by using electrical beamforming.

At this point, one of the main issues is how to obtain the best pointing direction. To do so it is important to know how the transmission channel works in all the different scenarios considered in FWA deployments, both at indoors and outdoors. Within 5G-AFIANCE project, mmWave channel characterization for outdoor to indoor scenarios will be carried out in order to characterizing the losses and the propagation characteristics [1].

This document describes the implementation carried out to automatize the channel measurements using a mechanical pointing system. This mechanical pointing system also can be used for reaching the best pointing direction indoors at mmW frequencies. Next section describes the first implementations of the

mechanical pointing system. Section 3 describes the automation system implemented to collect the channel measurements in an efficient way.

# 2 Mechanical Pointing System

The physical design of the mechanical pointing system is included in [1]. Figure 1 shows the implementation of the mechanical pointing system. This implementation is a good approximation of the mechanical pointing system that could be included in the design of the customer premises equipment (CPE) for indoor deployments.

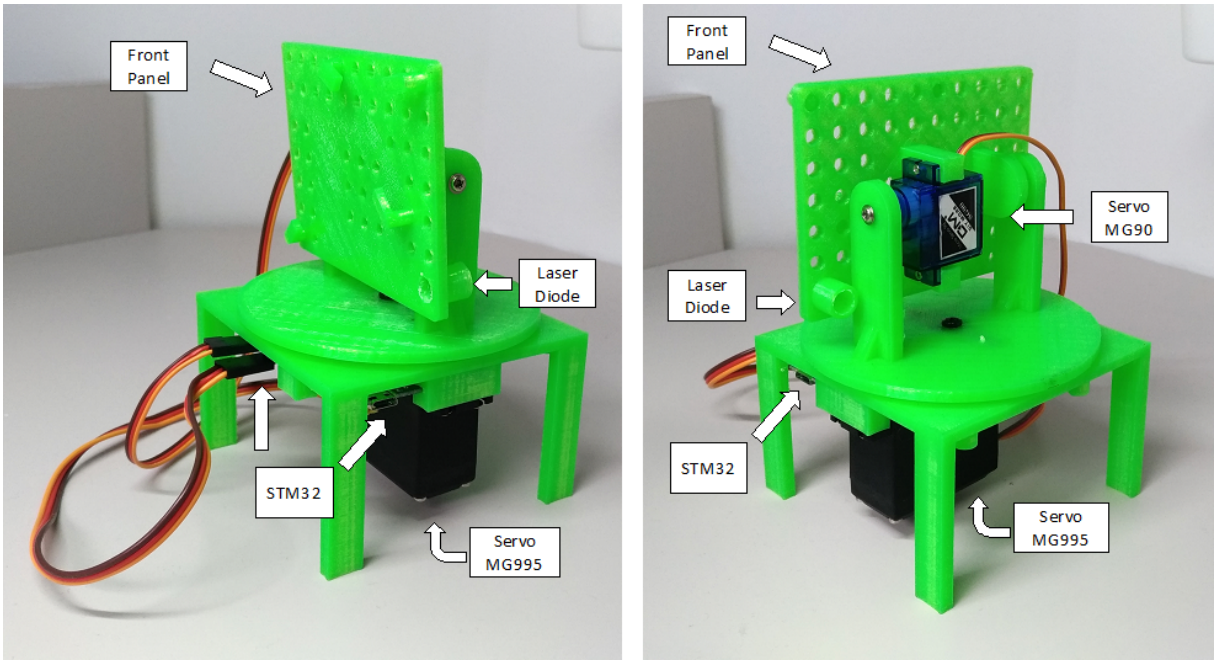


**Figure 1: Mechanical Pointing System.**

The pointing direction is controlled using an STM32 microcontroller and the software is developed in C programming language. Different pointing controlling software versions have been developed in order to evaluate and validate the mechanical pointing system.

- **Externally controlled version:** This version allows us to control the pointing direction via a serial connection to the STM32. This version has been developed to automatize the channel measurement process. Section 7, Annex I: Externally Controlled Version, includes the code implemented to automate the channel measurements.

- **Scanning version:** This version performs a pointing direction sweeping in order to find the best pointing direction. Using this version, it is possible to evaluate the time it takes to find the best pointing direction considering different parameters such as the sweep step at elevation and at azimuth. The time the servos need to set a new pointing direction is 3 ms per degree. Section 8, Annex II: Scanning Version, Annex II includes the code implemented to check the signal scanning at the UE side. Note that it is

possible to perform different scanning simulations running the scanning program only one time. At the first scanning attempt, the program uses an azimuthal and elevation sweep step of two degrees.

- **Beam Searching Version:** In [2] a method to automate the beam searching process was proposed. The method is based on using an on-off key (OOK) modulation to transmit the identifier of each of the different beams. At the transmitter side, the beam identifier is transmitted using the OOK modulation over a serial communication. At the receiver side, the signal should be amplified in order to be detected by the microcontroller, to do so a low noise amplifier (LNA) can be used. Once the signal is amplified it is rectifier using a Schottky diode. At this point, a sample and hold (S&H) circuit based on a MOSFET transistor can also be used to obtain the voltage of the received signal. In this case, the microcontroller can manage the sample and hold states to read the voltage received coming from the different beams. Section 9, Annex III: Beam Searching Version, includes the code implemented to carry out the autonomous beam searching.

# 3 Channel Measurements Automation

To optimize the channel measurements proposed in [1] different pieces of software have been developed. Figure 2 shows the setup to carry out the channel measurements.

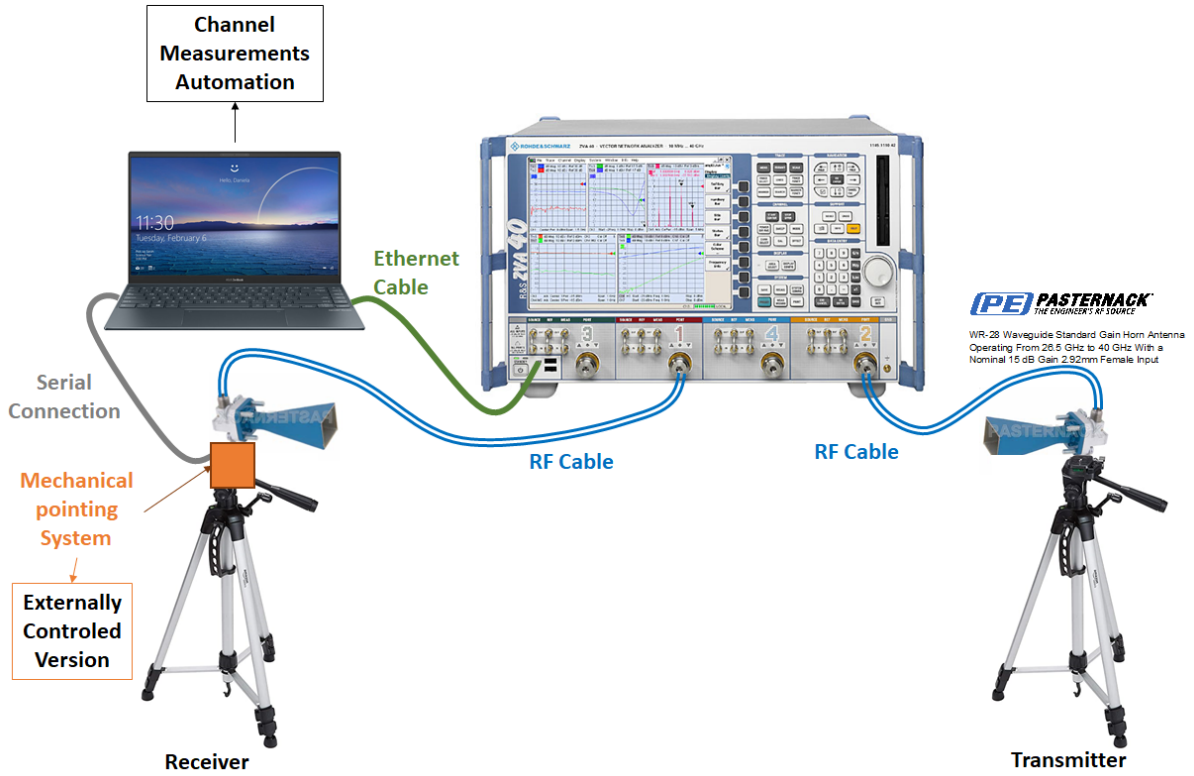


**Figure 2: Channel Measurement Setup**

There are two main blocks of code:

- Externally controlled version: This block of code as been explained in the previous section and it is included in section 7.

- **Channel Measurements Automation:** This block of code is written in Octave / MatLab programming language. This program is installed into the laptop that manages all the channel measurement process. This software:

    - Controls the mechanical pointing system and commands it.

    - Obtains the channel measurements coming from the vector network analyzer (VNA). In this case the software obtains the S-parameters. For us the most important S-parameter is the S21 because of this parameter gives us the relation between the transmitted signal and the received signal. Depending on the VNA the communication between the laptop and the VNA could be different [3][4][5]. Note that the VNA is controlled remotely using Standard Commands for Programmable Instruments (SCPI) programming language [6][7].

  Section 10, Annex IV: Channel Measurements Automation, includes the software implemented to automate the channel measurements. Note that the commands related to the communication between the laptop and the VNA could be different.

# 4 Conclusions

This document shows the different software implementations developed in 5GAFIANCE project in order to automate the channel measurement process and automate the beam searching process.

Three different programs have been written in C programming language to manage the mechanical pointing system in different ways. The first program allows us to manage the mechanical pointing system remotely. The second program allows us to apply different scanning steps and evaluate the time spent on that task. And the third program allows us to automate a beam searching process.

On the other side, in order to automate the channel measurements, a program written in Octave / MatLab code has been implemented. This program implements the main part of the channel measurements automation process. It manages both the mechanical pointing system and the vector network analyzer used for measuring the radio channel.

# 5 References


| | |
|---|---|
| [1] | Deliverable D6, “Beam Control Design”, 5GAFIANCE project, January 2021. |
| [2] | Deliverable D4, “Requirements for the Optimized FWA 5G Antenna”, February 2020. |
| [3] | Application Note, “Using MATLAB® for Remote Control and Data Capture with R&S Spectrum and Network Analyzers”, Rohde and Schwarz, January 2002. |
| [4] | Application Note, “How to use Rohde & Schwarz Instruments in MATLAB”, Rohde and Schwarz, December 2017. |
| [5] | Operating Manual, “R&S®ZVA / R&S®ZVB / R&S®ZVT Vector Network Analyzers”, Rohde and Schwarz, June 2020. |
| [6] | Standard Commands for Programmable Instruments (SCPI) Specification, European SCPI Consortium, 1999. |
| [7] | https://www.rohde-schwarz.com/fi/faq/all-four-s-parameter-remote-set-up-faq_78704-30474.html |

# 6 Acronyms

| | |
|---|---|
| CPE | Customer Premises Equipment |
| FWA | Fixed Wireless Access |
| LNA | Low Noise Amplifier |
| mmW | Millimeter Wave |
| OOK | On-Off Keying |
| S&H | Sample and Hold |
| SCPI | Standard Commands for Programmable Instruments |
| UE | User Equipment |
| VNA | Vector Network Analyzer |

# 7 Annex I: Externally Controlled Version

```
#include <Servo.h>

// DEFINE THE VARIABLES
Servo myServoAzimuth;       // create servo object to control the servo in azimuthal plane
Servo myServoElevation;     // create servo object to control the servo in elevation plane
int posAzimuth = 0;         // variable to store the servo position, initially at 0 degrees
int posElevation = 0;       // variable to store the servo position, initially at 0 degress
int cycle = 0;              // variable to store the number of steps accumulated in the process

// INITIAL SETUP
void setup() {

  pinMode(PB13, OUTPUT);            // IR LED input (LED Receiver)
  digitalWrite(PB13, LOW);          // Led initially off

  myServoAzimuth.attach(PB9);       // attaches the servo on pin 9 to the servo object
  myServoElevation.attach(PB8);     // attaches the elevation servo on pin 8 to the servo object
  Serial.begin(115200);             // Serial port to check the output of the program
  delay(1000);
  Serial.println("Preparing Servo, please wait");

  // Set the initial orientation
  myServoAzimuth.write(posAzimuth);
  myServoElevation.write(posElevation);

  // Test the azimuthal servo
  posAzimuth = 90;
  delay(1000);
  Serial.println("Testint the servo in the azimuthal orientation");
  myServoAzimuth.write(posAzimuth);
  delay(1000);
  for (posAzimuth = 0; posAzimuth <= 180; posAzimuth += 1) {  // azimuth test forward
    // in steps of 1 degree
    myServoAzimuth.write(posAzimuth);       // tell servo to go to position in variable 'posAzimuth'
    delay(50);                              // waits 50ms for the servo to reach the position
  }
  delay(1000);
  for (posAzimuth = 180; posAzimuth >= 0; posAzimuth -= 1) {  // azimuth test reverse
    myServoAzimuth.write(posAzimuth);       // tell servo to go to position in variable 'posAzimuth'
    delay(50);                              // waits 50ms for the servo to reach the position
    delay(50);                              // waits 50ms for the servo to reach the position
  }

  // Test the elevational servo
  posElevation = 90;
  delay(1000);
  Serial.println("Testint the servo in the elevational orientation");
  myServoElevation.write(posElevation);
  delay(1000);
  for (posElevation = 0; posElevation <= 180; posElevation += 1) {  // elevation test forward
    myServoElevation.write(posElevation);   // tell servo to go to position in variable 'posElevation'
    delay(50);                              // waits 50ms for the servo to reach the position
  }
  delay(1000);
  for (posElevation = 180; posElevation >= 0; posElevation -= 1) {  // elevation test reverse
    myServoElevation.write(posElevation);   // tell servo to go to position in variable 'posElevation'
    delay(50);                              // waits 50ms for the servo to reach the position
  }

  delay(1000);
  Serial.println("Intial Test Completed.");
  Serial.flush();
} // Setup

void loop() {

  String orientaion = "180";// Variable to store the orientation introduced using the serial port
  int azi, til;             // Variables to store the azimut & titl values introduced using the serial port

  if (Serial.available()) {     // Check if there is info avariable at the serial port
    // Print input information
    Serial.print("New cycle: ");
    Serial.print(cycle);
    Serial.print(" -> ");
    cycle++;                                    // Increase the cycle counter
```

```
    orientaion = Serial.readString();       // Read the incoming data as string
    Serial.println(orientation);            // Print the input coming from the serial port
    // Loop for obtaining the azimut and elevation values
    for (int i = 0; i < orientaion.length(); i++) {
      // Find the comma that separates the azimut and elevation values
      if (orientaion.substring(i, i + 1) == ",") {
        azi = orientaion.substring(0, i).toInt();   // store the azimut value
        til = orientaion.substring(i + 1).toInt();  // store the tilt value
        break;
      }
    }

    if (til > 90) {
      til = 90;         // trunk the tilt value if the value introduced is >90°
    }

    if (azi > 180) {    // Reset the azimuthal value when it is >180°
      azi = azi - 180;              // new azimuthal value
      til = 180 - til;              // new tilt value
    }

    // Set the orientation introduced using the serial port to the servos
    myServoAzimuth.write(azi);      // Tell servo to go to position in variable 'azi'
    myServoElevation.write(til);    // Tell servo to go to position in variable 'til'

    digitalWrite(PB13, HIGH);       // Turn on the led
    delay(3000);                    // wait 5 seconds
    digitalWrite(PB13, LOW);        // Turn off the led

  } // Serial Port Availability Checking
} // Loop
```

# 8 Annex II: Scanning Version

```
#include <Servo.h>

#define     MAX_ELEV_RANGE  20              // maximim value for elevation
#define     MAX_AZIM_RANGE  360             // maximim value for azimuth

Servo myServoAzimuth;       // create servo object to control the servo in azimuthal plane
Servo myServoElevation;     // create servo object to control the servo in elevation plane
int posAzimuth = 0;         // variable to store the servo position, initially at 0 degrees
int posElevation = 0;       // variable to store the servo position, initially at 0 degress
int cycle = 0;              // variable to store the number of steps accumulated in the process
int elevStep = 2;           // variable to store the elevation step (Take care with RAM memory cost)
int azimStep = 2;           // variable to store the elevation step (Take care with RAM memory cost)
unsigned long initTime, endTime, timeDuration;  // time related variables
int a = 0;                  // end loop variable

void setup() {

  myServoAzimuth.attach(PB9);       // attaches the servo on pin 9 to the servo object
  myServoElevation.attach(PB8);     // attaches the elevation servo on pin 8 to the servo object
  Serial.begin(115200);             // Serial port to check the output of the program
  delay(1000);
  Serial.println("Preparing Servo, please wait");

  // Set the initial orientation
  myServoAzimuth.write(posAzimuth);
  myServoElevation.write(posElevation);

  // Test the azimuthal servo
  posAzimuth = 90;
  delay(1000);
  Serial.println("Testint the servo in the azimuthal orientation");
  myServoAzimuth.write(posAzimuth);
  delay(1000);
  for (posAzimuth = 0; posAzimuth <= 180; posAzimuth += 1) {  // 0° => 180° in steps of 1°
    // in steps of 1 degree
    myServoAzimuth.write(posAzimuth);// tell servo to go to position in variable 'posAzimuth'
    delay(50);                       // waits 50ms for the servo to reach the position
  }
  delay(1000);
  for (posAzimuth = 180; posAzimuth >= 0; posAzimuth -= 1) {  // 0° => 180° in steps of 1°
    myServoAzimuth.write(posAzimuth); // tell servo to go to position in variable 'posAzimuth'
    delay(50);                        // waits 50ms for the servo to reach the position
  }

  // Test the elevational servo
  posElevation = 90;
  delay(1000);
  Serial.println("Testint the servo in the elevational orientation");
  myServoElevation.write(posElevation);
  delay(1000);
  for (posElevation = 0; posElevation <= 180; posElevation += 1) {  // 0° => 180° in steps of 1°
    myServoElevation.write(posElevation); // tell servo to go to position in variable 'posElevation'
    delay(50);                            // waits 50ms for the servo to reach the position
  }
  delay(1000);
  for (posElevation = 180; posElevation >= 0; posElevation -= 1) {  // 0° => 180° in steps of 1°
    myServoElevation.write(posElevation);   // tell servo to go to position in variable 'pos'
    delay(50);                              // waits 50ms for the servo to reach the position
  }

  delay(1000);
  Serial.println("Intial Test Completed.");
  Serial.flush();
}

void loop() {

  String simConfig = "180"; // Variable to store the configuration of the simulation

  // check if the loop has been already run
  if (a == 0){

    delay(1000);
    Serial.println("Starting Searching Process.");
    Serial.flush();
    initTime = millis();            // store the timestamp for at the beginning of the scanning process
```

```
    for (posAzimuth = 0; posAzimuth < MAX_AZIM_RANGE; posAzimuth += azimStep) {  // Azimuth
      delay(50);

      if (posAzimuth <= 180) {              // check the azimuth value
        myServoAzimuth.write(posAzimuth);       // set azimuth value
        for (posElevation = 0; posElevation < MAX_ELEV_RANGE; posElevation += elevStep) { // Elevation
          myServoElevation.write(posElevation);   // set elevation value
          delay(50);                              // wait 50ms for the servo to reach the position
        }
      } else {
        myServoAzimuth.write(posAzimuth - 180);   // set azimuth value
        for (posElevation = 0; posElevation < MAX_ELEV_RANGE; posElevation += elevStep) {  // elevation
          myServoElevation.write(180 - posElevation);   // set elevation value
          delay(50);                                        // waits 50ms for the servo to reach the position
        }
      }
      delay(50);                                          // waits 50ms for the servo to reach the position
    }

    endTime = millis();                     // store the timestamp for at the beginning of the scanning
    process
    timeDuration = endTime - initTime;  // obtain the time duration of the scanning process

    // print the time duration of the scanning process
    Serial.print("Duration Time = ");
    Serial.println(timeDuration);
    Serial.print("The azimuth and elevation steps configured are: [azimuth,elevation]");
    Serial.print(azimStep);
    Serial.print(",");
    Serial.println(elevStep);
    Serial.println("Process has ended");
    a = 1;                                // indicate the scanning has ended

  } else {
    delay(10000);
    Serial.println("Enter a combination of azimuth and elevation step values separated by a comma.");
    Serial.println("Example: [5,10]");

    if (Serial.available()) {     // Check if there is info avariable at the serial port
      // Print input information
      Serial.print("--- New simulation --- Simulation number --- ");
      Serial.print(cycle);
      Serial.println("---");
      cycle++;                                            // Increase the cycle counter

      simConfig = Serial.readString();              // Read the incoming data as string
      Serial.println(simConfig);                                  // Print the input coming from the serial
      port
      // Loop for obtaining the azimut and elevation values
      for (int i = 0; i < simConfig.length(); i++) {
        // Find the comma that separates the azimut and elevation values
        if (simConfig.substring(i, i + 1) == ",") {
          azimStep = simConfig.substring(0, i).toInt();     // store the azimuth step value
          elevStep = simConfig.substring(i + 1).toInt();    // store the tilt step value
          break;
        }
      }
      a = 0;    // indicate the controller to resimulate
    } // serial communication checking
  } // else statement end
} // loop
```

# 9 Annex III: Beam Searching Version

```
#include <Servo.h>

byte        beamID;                         // Beam ID of the received Beam
#define     MAX_ELEV_RANGE  20              // maximim value for elevation
#define     MAX_AZIM_RANGE  360             // maximim value for azimuth
#define     ELEV_STEP       2               // Elevation Step (Take care with RAM memory cost)
#define     AZIM_STEP       2               // Azimuth Step (Take care with RAM memory cost)
uint16_t    recvEnergy[MAX_AZIM_RANGE / AZIM_STEP][MAX_ELEV_RANGE / ELEV_STEP];
byte        recvBeam[MAX_AZIM_RANGE / AZIM_STEP][MAX_ELEV_RANGE / ELEV_STEP];
int         bit7, bit6, bit5, bit4, bit3, bit2, bit1, bit0, auxCount;
int         bestAzim = 0, bestElev = 0, bestBeamID = 0, bestPower = 0;
int a = 0, b = 0;                           // end loop variable (a), blink (b)

Servo myServoAzimuth;       // create servo object to control the servo in azimuthal plane
Servo myServoElevation;     // create servo object to control the servo in elevation plane
int posAzimuth = 0;         // variable to store the servo position, initially at 0 degrees
int posElevation = 0;       // variable to store the servo position, initially at 0 degress
int cycle = 0;              // variable to store the number of steps accumulated in the process

void setup() {

  pinMode(PA1, INPUT);              // Received Signal Voltage

  pinMode(PB13, OUTPUT);            // IR LED input (LED Receiver)
  digitalWrite(PB13, LOW);          // Led initially off

  pinMode(PB12, OUTPUT);            // MOSFET Controller Port
  digitalWrite(PB12, LOW);          // MOSFET (S&H) initially opened (Hold State)

  Serial1.begin(115200);            // Serial communication used for obtaining the beam ID

  myServoAzimuth.attach(PB9);       // attaches the servo on pin 9 to the servo object
  myServoElevation.attach(PB8);     // attaches the elevation servo on pin 8 to the servo object
  Serial.begin(115200);             // Serial port to check the output of the program
  delay(1000);
  Serial.println("Preparing Servo, please wait");

  // Set the initial orientation
  myServoAzimuth.write(posAzimuth);
  myServoElevation.write(posElevation);

  // Test the azimuthal servo
  posAzimuth = 90;
  delay(1000);
  Serial.println("Testint the servo in the azimuthal orientation");
  myServoAzimuth.write(posAzimuth);
  delay(1000);
  for (posAzimuth = 0; posAzimuth <= 180; posAzimuth += 1) {  // 0° => 180° in steps of 1°
    myServoAzimuth.write(posAzimuth);       // tell servo to go to position in variable 'posAzimuth'
    delay(50);                              // waits 50ms for the servo to reach the position
  }
  delay(1000);
  for (posAzimuth = 180; posAzimuth >= 0; posAzimuth -= 1) {  // 0° => 180° in steps of 1°
    myServoAzimuth.write(posAzimuth);       // tell servo to go to position in variable 'pos'
    delay(50);                              // waits 50ms for the servo to reach the position
  }

  // Test the elevational servo
  posElevation = 90;
  delay(1000);
  Serial.println("Testint the servo in the elevational orientation");
  myServoElevation.write(posElevation);
  delay(1000);
  for (posElevation = 0; posElevation <= 180; posElevation += 1) {  // 0° => 180°  in steps of 1 °
    myServoElevation.write(posElevation);   // tell servo to go to position in variable 'posElevation'
    delay(50);                              // waits 50ms for the servo to reach the position
  }
  delay(1000);
  for (posElevation = 180; posElevation >= 0; posElevation -= 1) {  // 0° => 180°  in steps of 1 °
    myServoElevation.write(posElevation);   // tell servo to go to position in variable 'posElevation'
    delay(50);                              // waits 50ms for the servo to reach the position
  }

  delay(1000);
  Serial.println("Intial Test Completed.");
  Serial.flush();
}
```

```
void loop() {

  // check if the loop has been already run
  if (a == 0){
    delay(1000);
    Serial.println("Starting Searching Process.");
    Serial.flush();
    delay(50);

    for (posAzimuth = 0; posAzimuth < MAX_AZIM_RANGE; posAzimuth += AZIM_STEP) { // Azimuth
      if (posAzimuth <= 180) {
        myServoAzimuth.write(posAzimuth);       // tell servo to go to position in variable 'posAzimuth'
        for (posElevation = 0; posElevation < MAX_ELEV_RANGE; posElevation += ELEV_STEP) {  // Elevation
          myServoElevation.write(posElevation); // tell servo to go to position in variable 'posElevation'
          delay(50);                            // waits 50ms for the servo to reach the position.
          if (Serial1.available() > 0) {          // check it there is any value at the serial port1
            beamID = Serial1.read();            // Read the BeamID value
            // Obtaine each bit of the byte word
            bit7 = bitRead(beamID, 0);
            bit6 = bitRead(beamID, 1);
            bit5 = bitRead(beamID, 2);
            bit4 = bitRead(beamID, 3);
            bit3 = bitRead(beamID, 4);
            bit2 = bitRead(beamID, 5);
            bit1 = bitRead(beamID, 6);
            bit0 = bitRead(beamID, 7);
            // The word must have the same number of ones and zeros
            auxCount = bit7 + bit6 + bit5 + bit4 + bit3 + bit2 + bit1 + bit0;
            if (auxCount == 4) {
              recvBeam[posAzimuth][posElevation] = beamID;            // Set the beam ID matrix
              digitalWrite(PB12, HIGH);                               // MOSFET (S&H) closed (Sample State)
              delayMicroseconds(80);                                  // 80 usec delay for 10 symbols
              digitalWrite(PB12, LOW);                                // MOSFET (S&H) opened (Hold State)
              recvEnergy[posAzimuth][posElevation] = analogRead(PA1); // Obtain the power value
              if (recvEnergy[posAzimuth][posElevation] > bestPower) { // update best orientation
                bestAzim = posAzimuth;                                  // best azimuth value update
                bestElev = posElevation;                                // best elevation value update
                bestBeamID = beamID;                                  // best beam ID value update
                bestPower = recvEnergy[posAzimuth][posElevation];       // best light power value update
              }
            }
          }
        }
      } else {
        myServoAzimuth.write(posAzimuth - 180);     // tell servo to go to position in variable 'posAzimuth'
        for (posElevation = 0; posElevation < MAX_ELEV_RANGE; posElevation += ELEV_STEP) {
          myServoElevation.write(180 - posElevation);   // set elevation value on the servo
          delay(50);                                    // waits 50ms for the servo to reach the position
          beamID = Serial1.read();                      // Read the BeamID value
          // Obtaine each bit of the byte word
          bit7 = bitRead(beamID, 0);
          bit6 = bitRead(beamID, 1);
          bit5 = bitRead(beamID, 2);
          bit4 = bitRead(beamID, 3);
          bit3 = bitRead(beamID, 4);
          bit2 = bitRead(beamID, 5);
          bit1 = bitRead(beamID, 6);
          bit0 = bitRead(beamID, 7);
          // The word must have the same number of ones and zeros
          auxCount = bit7 + bit6 + bit5 + bit4 + bit3 + bit2 + bit1 + bit0;
          if (auxCount == 4) {
            recvBeam[posAzimuth][posElevation] = beamID;
            digitalWrite(PB12, HIGH);                                 // MOSFET (S&H) closed (Sample State)
            delayMicroseconds(80);                                    // 80 usec delay for 10 symbols
            digitalWrite(PB12, LOW);                                  // MOSFET (S&H) opened (Hold State)
            recvEnergy[posAzimuth][posElevation] = analogRead(PA1); // Obtain the power value
            if (recvEnergy[posAzimuth][posElevation] > bestPower) { // update best orientation
              bestAzim = posAzimuth;                                  // best azimuth value update
              bestElev = posElevation;                                // best elevation value update
              bestBeamID = beamID;                                    // best beam ID value update
              bestPower = recvEnergy[posAzimuth][posElevation];       // best light power value update
            }
          }
        }
      }
    }
```

```
    delay(50);        // waits 50ms for the servo to reach the position.

    // Point to the best orientation
    if (bestAzim <= 180) {
      myServoAzimuth.write(posAzimuth);
      myServoElevation.write(posElevation);
    } else {
      myServoAzimuth.write(posAzimuth - 180);
      myServoElevation.write(180 - posElevation);
    }

    a = 1;                      // indicate the scanning has ended
    digitalWrite(PB13, HIGH);   // Turn the led On
    delay(1000);

  } else {                  // make the led blink
    if (b == 0) {
      digitalWrite(PB13, LOW);  // Turn the led off
      b = 1;
      delay(1000);
    } else {
      digitalWrite(PB13, HIGH); // Turn the led off
      b = 0;
      delay(1000);
    }
  }
}
```

# 10 Annex IV: Channel Measurements Automation

```
clc;clear;

% IMPORTANT NOTE:
% This code has been written using Octave. If you use MatLab software instead of
% octave you must adapt this code to Matlab by replacing the hashtag (%) by the
% percentage symbol (%). Other changes could be also needed.

% load package for managing the serial connection
pkg load instrument-control

%% PARAMETERS DEFINITION

% Azimuth configuration
az_ini = 0;
az_step = 1;
az_end = 360;
% Elevation configuration
el_ini = 0;
el_step = 1;
el_end = 360;

% Matrix to store the different measures
az_meas = floor((az_end - az_ini) / az_step);
el_meas = floor((el_end - el_ini) / el_step);


% Depending on if we are going to store one measurement per orientation or
% we are going to store more than one measurement per orientation we can use
% different techniques.

% Using Cell Array: It allows is to manage different measurements at each
% orientation individually. We can have different number of measurements per
% orientation.
channel_matrix_cell = cell(az_meas, el_meas);


% If the number of measurements per orientation is going to be fixed, it could
% be better to work with matrices because of it is faster than working with
% cells
n_meas = 10;                    % number of measurements
channel_matrix = zeros(az_meas, el_meas, n_meas);

% IP address of the VNA
% Note: Please confirm the IP address before running this program
Ip_Addr = "89.10.66.41";

% Open serial port
% NOTE: the port name could be different depending on the PC used. Please,
% confirm the serial port name before running this code.
s1 = serial("COM3");

% Introduce 1 second delay for configuration.
pause(1);
```

```
% Serial Port Settings
set(s1, "baudrate", 115200);     % Baudrate. Currently 115200
set(s1, "bytesize", 8);          % Bytesize
set(s1, "parity", "n");          % Parity bit
set(s1, "stopbits", 1);          % Stop Bit
%%set(s1, "timeout", 50);          % timeout: In this case timeout is not needed

% Flush input and output buffers
srl_flush(s1);

% Open connection to the VNA with IP address = Ip_Addr
ud=mexrsib('ibfind', Ip_Addr);

% check the connection with the VNA
if ud > 0
  % Init device
  mexrsib('ibwrt',ud,'*RST;*CLS');

  % Identify the instrument: Manufacturer + Model + Firmware level
  mexrsib('ibwrt',ud,'*IDN?');
  IdStr = mexrsib('ilrd',ud,100);

  % Turn display update ON
  mexrsib('ibwrt',ud,'SYST:DISP:UPD ON');

  % Set screen for S11
  mexrsib('ibwrt',ud,'CALC1:PAR:SDEF TRC1, S11');
  mexrsib('ibwrt',ud,'CALC1:FORM SMITH');
  mexrsib('ibwrt',ud,'DISP:WIND1:STAT ON');
  mexrsib('ibwrt',ud,'DISP:WIND1:TRAC:FEED TRC1');

  % Set screen for S12
  mexrsib('ibwrt',ud,'CALC1:PAR:SDEF TRC2, S12');
  mexrsib('ibwrt',ud,'CALC1:FORM SMITH');
  mexrsib('ibwrt',ud,'DISP:WIND2:STAT ON');
  mexrsib('ibwrt',ud,'DISP:WIND2:TRAC:FEED TRC2');

  % Set screen for S11
  mexrsib('ibwrt',ud,'CALC1:PAR:SDEF TRC3, S21');
  mexrsib('ibwrt',ud,'CALC1:FORM SMITH');
  mexrsib('ibwrt',ud,'DISP:WIND3:STAT ON');
  mexrsib('ibwrt',ud,'DISP:WIND3:TRAC:FEED TRC3');

  % Set screen for S11
  mexrsib('ibwrt',ud,'CALC1:PAR:SDEF TRC4, S22');
  mexrsib('ibwrt',ud,'CALC1:FORM SMITH');
  mexrsib('ibwrt',ud,'DISP:WIND4:STAT ON');
  mexrsib('ibwrt',ud,'DISP:WIND4:TRAC:FEED TRC4');

  % Select single sweep
  mexrsib('ibwrt',ud,'INIT:CONT OFF');

else
  error ("The VNA is not connected");
end
```

```
% Loops for scanning different pointing directions
for az = az_ini:az_step:az_end      % azimut
  for el = el_ini:el_step:el_end    % elevation
    % string to write into the serial connection
    aux_str = [num2str(az) "," num2str(el)];
    % write a string message throught the serial connection
    w_bytes = srl_write(s1, aux_str);
    if length(aux_str) == w_bytes
      printf("The current pointing direction is:\n");
      printf("%d grades in azimuthn\n%d grades in elevation.\n\n", az, el);
    else
      printf("ERROR: An error has occurred.");
      break
    end
        % Perform one sweep and wait for operation to complete
        mexrsib('ibwrt',ud,'INIT;*OPC?');
        y=mexrsib('ilrd',ud,1);

        % Obtain all the parameters at once
        mexrsib('ibwrt',ud,'CALC1:DATA:ALL? SDAT');

        %%% COMPLETE THE READING PROCESS

    pause(1);
  end
end

% Current data time
date_and_time = strftime("%Y%m%d_%H%M%S", localtime(time()));
% Save the results and the configuration
save(date_and_time);
```

# 11 Annex V: Final Channel Measurement Automation

Once the mechanical pointing system was done, it is necessary to assemble it with the measurement setup, that is, the VNA with cables and the horn antennas.

The final mechanical pointing system include a horn antenna holder, as shown Fig. 3 and higher arms to prevent the cables from obstructing rotational movement.

The VNA will be connected to the horn antenna through coaxial cables with SMA connectors.

The rotative antenna has two rotation axes made by two motors. These motors are commanded by an Arduino UNO (initially an STM32 was going to be used), which gives the necessary instructions to the motors so that the antenna points in a certain direction.

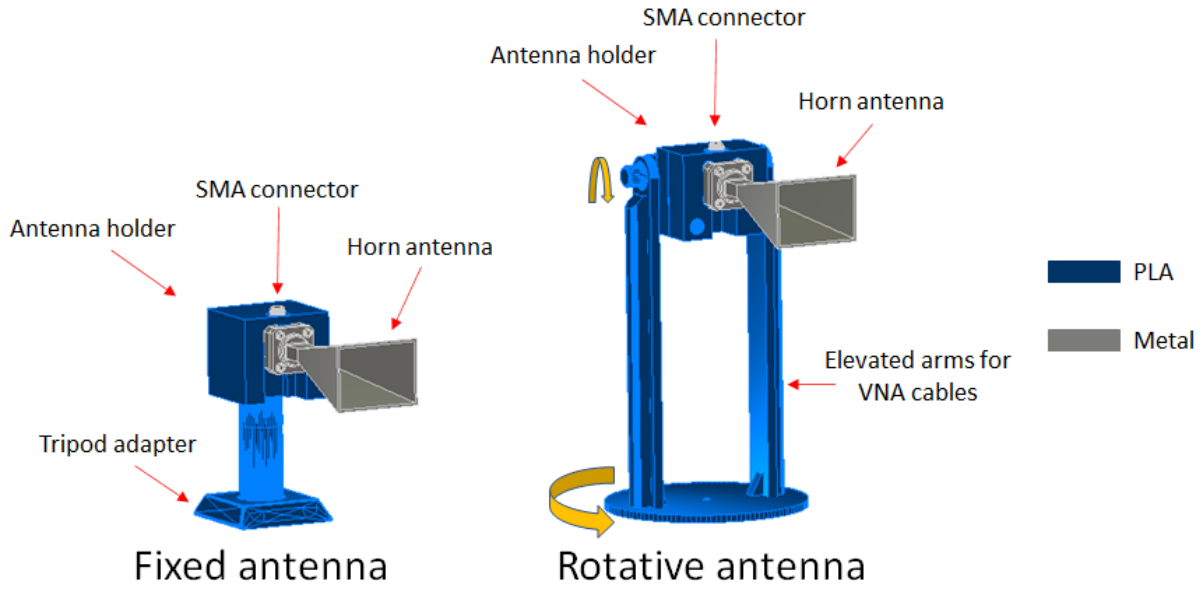


**Figure 3: Mechanical antenna holders and rotative system.**

A program was done in MatLAB to measure the transmission coefficient (S12) through two horn antennas in an automatized way. At each time, the pointing system modifies the angular position of the rotative horn antenna. This MatLAB code is explained in the block diagram of Fig.4.

The following are the most relevant variables of this algorithm:

- Starting frequency: the starting frequency value of the bandwidth.
- Final frequency: the stop final frequency value of the bandwidth.

- Number of points: total measured frequency point inside the bandwidth.
- Number of measurements: total of measurement done per angular position. A higher measurement number decrease the noise.
- Azimuthal angular step: azimuthal angular difference between two consecutive measurements.
- Elevation angular step: elevation angular difference between two consecutive measurements.

The azimuthal and elevation angular steps define the numbers of total measures in the half sphere, that is, the number of cycles to be perform.

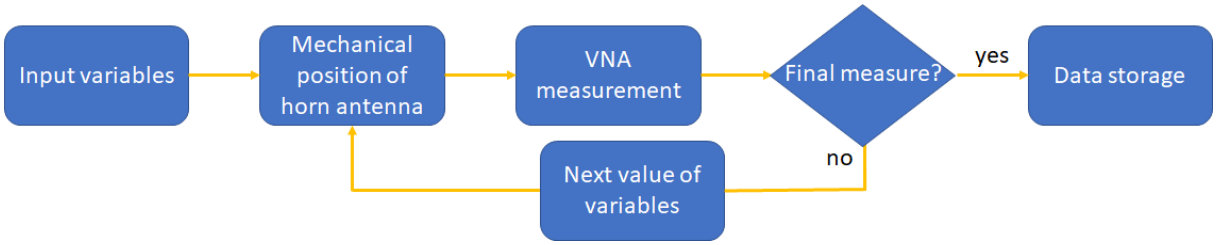


**Figure 4: Block diagram of data measuring.**