AUTOMATION OF A VECTOR NETWORK ANALYZER AND A STEPPER MOTOR USING THE BEAD PULL METHOD AS AN APPLICATION EXAMPLE

Bachelor Thesis in Physics

by

Nils Deimann

Presented to
Physics Institute III A
RWTH Aachen University

February 2023

First assessor and main supervisor:	Prof. Dr. A. Schmidt
Secondary assessor:	Prof. Dr. M. Erdmann
Supervisor:	Dr. Erdem Öz

Acknowledgments

I would like to thank Prof. Dr. Alexander Schmidt for the opportunity to write this work as well as for the first insights into the scientific work. I also thank Dr. Erdem Öz for the fruitful discussions and constant help with my questions. In general, the working atmosphere was consistently very pleasant. Therefore, I would also like to thank the rest of the research group. Finally, I would like to thank my very good friend Lukas, who always supported and motivated me during my work.

Abstract

Axions are hypothetical particles that could solve many problems in modern physics, such as the dark matter problem or the strong CP problem. The MADMAX project is trying to find this particle. Automated codes could help to research more efficiently. The goal of this work is to perform a simple measurement using a Python script. Using the example of motors and a network analyzer, an attempt was made to establish a connection and then to check it.

Contents

1 Introduction

1.1 The strong CP problem, the axion and dark matter

CP-symmetry describes the conservation of the laws of physics, if the charge (C) and the parity (P) is interchanged. So if one look at a anti-particle in a location-inverted world, it should be acting like the normal particle in our world (if there is CP-symmetry). According to James Cronin and Val Fitch (Nobel Prize in physics 1980) CP-symmetry is violated. It is well-known to be broken through weak interactions. The standard model of particle physics implies that strong interactions should also violate CP-symmetry.

In quantum chromodynamics (QCD) two terms come within the regarding Lagrangian, if one considers a single massive quark[Wu97]. These two terms are CP-violating. First the mass term $\mathcal{L}_m$ and second the θ-term $\mathcal{L}_\theta$ with

$$\mathcal{L} = \mathcal{L}_{non-violating} + \mathcal{L}_m + \mathcal{L}_\theta \tag{1}$$

where

$$\mathcal{L}_m = -\bar{\psi}(me^{i\gamma_5\phi})\psi \tag{2}$$

and

$$\mathcal{L}_\theta = \theta G\tilde{G} \tag{3}$$

One can try to get rid of the CP-violating terms by performing a chiral transformation by α as the angle.

$$\psi' = e^{i\alpha\gamma_5/2}\psi \tag{4}$$

The transformation changes the two terms $\mathcal{L}_m$ and $\mathcal{L}_\theta$ by $\phi \to (\phi - \alpha)$ and $\theta \to (\theta + \alpha)$. But $\phi + \theta$ will not change under chiral rotation, because:

$$\phi + \theta \to (\phi - \alpha) + (\theta + \alpha) = \phi + \theta \tag{5}$$

So arbitrary how α is choosed, there is always a CP-violating term left. For example if one choose $\alpha = \phi$ all the CP-violation comes from the θ-term.

Today there are upper bounds on the electric dipole moment of the neutron, which are experimentally verified [Bak+06]. These upper bounds requires $\bar{\theta} \leq 10^{-11}$, where $\bar{\theta}$ is the CP-violating angle in the Standard Model regarding to six quarks instead of one. Because $\bar{\theta}$ can take values between 0 and 2π it is not clear why $\bar{\theta}$ must be so small. This problem is called the strong CP problem.

To solve the strong CP problem Roberto Peccei and Helen Quinn proposed the so-called Peccei–Quinn theory in 1977 [PQ77b] [PQ77a]. In this theory a new symmetry and a new scalar field is introduced. At low energies the scalar field spontaneously breaks the symmetry, which leads to a new particle, known as the axion. The axion is a hypothetical elementary particle with no electrical charge and zero spin. A consequence of the Peccei-Quinn theory is a new term in the Lagrangian, which leads to the cancellation of the CP-violating θ-term. Therefore this theory solves the strong CP problem.

The axion is also a possible candidate for dark matter. Dark matter is a postulated form of matter, which is not visible, but interacts with gravity. One reason for postulating such form of matter is the movement of stars in galaxies. The observed orbital velocities of stars are higher in the outer regions of galaxies than expected [CS00]. There are more problems, dark matter could explain. In summary, this makes the search for the axion even more attractive. Therefore many physicists try to develop experiments to find the axion.

1.2 MADMAX

The Magnetized Disk and Mirror Axion eXperiment (MADMAX) is being developed to search for the axion. MADMAX especially searches in the mass range of $40 - 400\mu eV$, which is a highly attractive and well motivated region for dark matter axions [Cal+19]. It is a haloscope, which contains a large amount of dielectric disks ($\varepsilon \gg 1$), where a electromagnetic signal is sent through. MADMAX itself is based on the Primakoff effect. At the transition between the dielecrtic material and vacuum and within the magnetic field ($B_e = 10T$) the axion produces electromagnetic waves perpendicular to the surface of the dielectric material. Because the waves have such a small intensity, it is necessary to use many disks to boost the signal. The power P of the produced wave can be calculated by following:

$$\frac{P}{A} = \beta^2 \cdot \frac{P_0}{A} = 2.2 \cdot 10^{-27} \frac{W}{m^2} \beta^2 \left(\frac{B_e}{10T} \right)^2 C_{a\gamma}^2 \tag{6}$$

For the whole setup, the main interest is on the boost factor β^2, which compares the measured

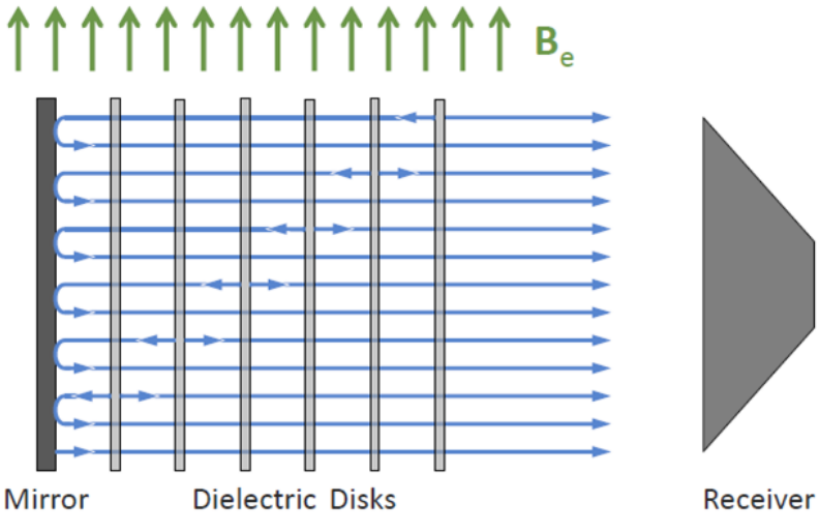

Figure 1: Haloscope with mirror and multiple dielectric disks; B_e represents a magnetic field, which is perpendicular to direction of the propagating wave

power P of the haloscope with the power P_0 if only the mirror. The boost factor is frequency dependent. To finally detect an axion signal the boost factor must be very well known. But the predicted power P_0 must be interpolated from the signal by knowing the boost. However, one would need to know the power of the axion signal generated at a surface in order to determine the amplification from the measured signal. Therefore the boost factor is one of the biggest issues right now on MADMAX. On website of the MADMAX-collaboration are more information (https://madmax.mpp.mpg.de/index.html).

1.3 Motivation

In order to perform MADMAX it still requires a lot of research and many smaller experiments regarding microwaves. Next to simulations, experimental approaches are very important. In order to be able to experiment more effectively and precisely in the future, it makes sense to automate experiments. The main aspect of this work is therefore to obtain a first fully code controlled measurement as a demonstration. For this the whole setup and the entire code must be explained.

2 Theory

2.1 Microwave technology

Microwaves are electromagnetic waves within a frequency range of $1 - 300GHz$. They are well known in everyday life, whether you encounter them by WiFi or by an actual microwave. Today there are much more applications to microwaves and therefore many companies are interested in such technologies.

A Vector Network Analyzer (VNA) helps to study the behavior of microwaves. A VNA is an instrument that can produce electromagnetic waves through multiple ports (N-port system) and measure the properties of the incoming waves, also through the same ports. Instead measuring only the amplitude of an incoming wave - like in a Scalar Network Analyzer (SNA) - the VNA is measuring both, amplitude and phase properties. Therefore it can reproduce the scattering parameters and thus the scattering matrix (S-matrix) of a system, which is quite important.

For a 2-port system the VNA produces two waves a_1 (port 1) and a_2 (port 2), which will be transmitted and reflected by some environment. Then two new waves b_1 and b_2, resulting from the incident waves, will be measured by the VNA.

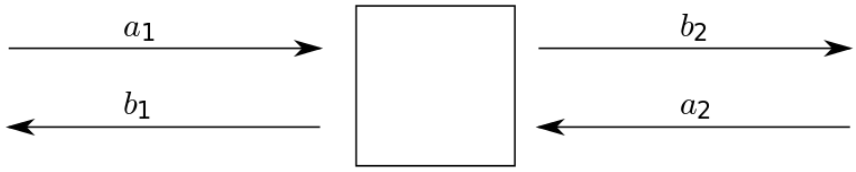

Figure 2: a_i and b_i are waves at port i; the box represents an environment, which mixes the properties of the wave; waves propagate in arrow direction

The relationship between the produced waves, the measured waves and the S-matrix is given by:

$$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} \tag{7}$$

In the following only S_{11} is relevant, because only one port was used. As the reflection coefficient, the ratio of the reflected wave b_1 and the incident wave a_1, it is given by $S_{11} = b_1/a_1$.

2.2 Bead pull method

In this chapter the bead pull method will be introduced. The idea of this method is to compare two measurements of standing waves. The first one is in vacuum and the second one comes with a bead, as a perturbation. More precisely, one measures the complex reflection coefficient S_{11}, which is discussed in chapter 2.1. The theory behind this is called non resonant perturbation theory and was first proposed by Steele [Ste66]. It allows the measurement of the electric field in the bead position by measuring the complex reflection coefficient at a known frequency [Mos+09]. The following formula explains the connection between the perturbed (unperturbed) reflection coefficient $S_{11,p}$ ($S_{11,u}$), the absolute value of the electric field E and the frequency ω, where P_{inc} is the average power of the incoming wave, i the imaginary unit and k_{ST} a geometric factor.

$$\Delta S_{11} = S_{11,p}(\omega) - S_{11,u}(\omega) = \frac{-i\omega k_{ST} E^2}{P_{inc}} \tag{8}$$

The absolute value of the electric field can then be calculated by following formula, after solving for E:

$$E = \sqrt{\left|\frac{P_{inc}}{k_{ST}}\right|} \cdot \sqrt{\left|\frac{S_{11,p}(\omega) - S_{11,u}(\omega)}{\omega}\right|} \tag{9}$$

3 Automazation

The main aspect of this thesis is to connect and program a stepper motor and a VNA (2.1) with a computer. This chapter will introduce and explain the necessary setup, software and hardware. Furthermore the code, which controls motor and VNA, will be explained.

3.1 Installing setup

The whole setup contains a computer, a VNA, a controller and two stepper motors. The motors and the controller come from the company Standa. Especially the controller with the model number 8SMC5-USB-B9-2 and the motors with the model numbers 8MT195/8MT295 will be considered. The goal is to send the proper commands to the controller, so the motors are doing the right thing. The connection with the controller is made via a USB cable. The VNA is a architecture by Keysight. The one used here has the model number N5224B. The VNA gets the commands directly from the computer via Wi-Fi. The following picture shows the setup:

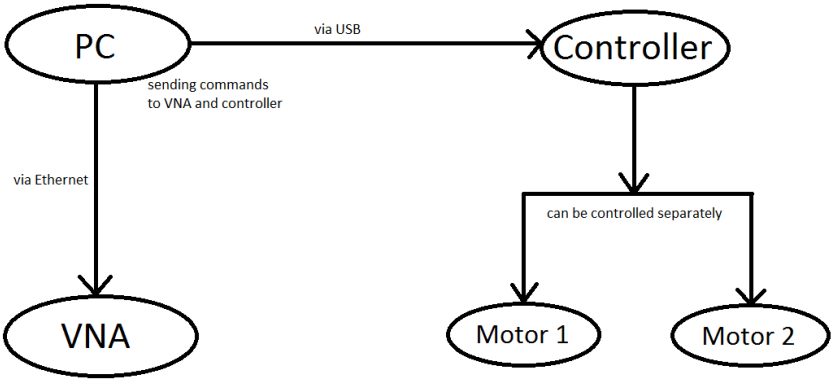

Figure 3: Connection between computer, VNA, controller and motors

Both are controlled by a python code. The code used here is based on two codes that have been combined and partially changed. The first can be found [On1]. This acts only on the VNA and provides the basis for the socket programming, which is explained in the course of this chapter. The second code deals with the connection of the engine and the libximc library. It is available for download with XiLab and the necessary libraries at [On2]. The resulting code and all needed files are at [On3]. The contents of this repository can be obtained using the command "git clone https://github.com/NilsDeimann/Automation-of-a-vector-network-analyzer-and-a-stepper-motor". However, the git program is required for this (`https://git-scm.com/`). The code can then be found in the path "...\XIMC\ximc-2.13.6\examples\test_Python\standardtest\testpython.py". After the necessary files are downloaded the process of how to get the whole setup done will be explained in the next two subsections.

3.1.1 Stepper motor

To control the motors, the libximc library is used. This library is recommended for usage because Standa itself used libximc to build XiLab. XiLab is the standard program to control the motors manually. Installing XiLab is therefore sufficient to get the drivers for the motors. The libraries must be added to the python path yet. This procedure will be explained in the code section 3.2.1.

To get a connection the USB cable must be plugged into the computer and the controller. To check the connection one can open the device manager whilst pressing Windows + R and entering "devmgmt.msc". The two ports for the two motors should then be found in the section "Ports (COM & LPT)". This will be important when one decides which motor should move (chapter 3.2.1). To finally check the connection, XiLab can be opened and the motor can be choosed. Then the motor should be able to be controlled. Furthermore it is important to check if, there are any other things, that uses libximc. If the code is executed, it could produce problems, therefore XiLab should be closed, while running the code.

3.1.2 VNA

To control the VNA the socket library is used. This library should be preinstalled. With socket it is possible to create a so-called socket, which is used to send and receive data, from one to another device. Especially sending data will be quite important, because commands must be sent to the VNA (chapter 3.2.2).

One have to connect the VNA with the local network via an Ethernet cable. To build a connection to the VNA, the IP of the VNA must be figured out. It can be found by typing "ipconfig" in the command prompt. How to use the IP to get a connection will be discussed in chapter 3.2.2.

3.2 Code

This chapter contains all information of the code itself. The code parts of the stepper motor and the VNA are described separately. At the end some examples of what are the possibilities with the code are shown.

3.2.1 Stepper motor

This chapter is about the part of the code, which is responsible for controlling the motors.

At first the necessary libraries must be implemented. They can be found, as explained in chapter 3.1.1, in the folder XIMC. Hence they must be added to the python path manually. The code down below (1) navigates in the directory (line 7-9), where the pyximc.py file is. This one will be added to the python path (line 10). Then depending on the system version (win32/win64) the libraries bindy.dll, xiwrapper.dll and libximc.dll will be also added to the python path (line 16-23). Those steps are necessary for the code to find the libraries and use them.

```
1  import os
2  import sys
3  import platform
4
5  # For correct usage of the library libximc,
6  # you need to add the file pyximc.py wrapper with the structures of the library to
   ↪  python path.
7  cur_dir = os.path.abspath(os.path.dirname(__file__)) # Specifies the current
   ↪  directory.
8  ximc_dir = os.path.join(cur_dir, "..", "..", "..", "ximc") # Formation of the
   ↪  directory name with all dependencies. The dependencies for the examples are
   ↪  located in the ximc directory.
9  ximc_package_dir = os.path.join(ximc_dir, "crossplatform", "wrappers", "python") #
   ↪  Formation of the directory name with python dependencies.
10 sys.path.append(ximc_package_dir)  # add pyximc.py wrapper to python path
11
12 # Depending on your version of Windows, add the path to the required DLLs to the
   ↪  environment variable
13 # bindy.dll
14 # libximc.dll
15 # xiwrapper.dll
16 if platform.system() == "Windows":
17     # Determining the directory with dependencies for windows depending on the bit
       ↪  depth.
18     arch_dir = "win64" if "64" in platform.architecture()[0] else "win32" #
19     libdir = os.path.join(ximc_dir, arch_dir)
20     if sys.version_info >= (3,8):
21         os.add_dll_directory(libdir)
22     else:
23         os.environ["Path"] = libdir + ";" + os.environ["Path"] # add dll path into an
           ↪  environment variable
```

Listing 1: adding pyximc.py and necessary libraries to the python path

After adding the files to the python path, they can be imported by following.

```
1 try:
2     from pyximc import *
```

Listing 2: library is imported

Afterwards the code searches for devices, connected to the computer. Down below (3) in line 2-5 the code enumerates all found connections and saves them as "devenum". It is also possible to broadcast devices manually with giving the parameter "enum_hints". In this scenario, this option is not used because the devices are found automatically. Then in line 10 the motor can be chosen out of all devices from the enumeration mentioned above. It is done by giving the function

"get_device_name()" an integer as an index of the enumeration. At last the ID of the motor can be saved with the function "open_device()" seen in line 12. This will be important when one choose, which motor should move or generally operate. To use multiple motors simultaneously, line 10 and 12 can be duplicated and saved as another variables. These variables are passed when instructions are executed so that the code knows which motor this instruction affects. The functions operates on the variable "lib". This variable points to the loaded library "pyximc" (2).

```
1  # This is device search and enumeration with probing. It gives more information about
   ↪  devices.
2  probe_flags = EnumerateFlags.ENUMERATE_PROBE + EnumerateFlags.ENUMERATE_NETWORK
3  enum_hints = b"addr="
4  # enum_hints = b"addr=" # Use this hint string for broadcast enumerate
5  devenum = lib.enumerate_devices(probe_flags, enum_hints)
6  print("Device enum handle: " + repr(devenum))
7  print("Device enum handle type: " + repr(type(devenum)))
8  dev_count = lib.get_device_count(devenum)
9  print("Device count: " + repr(dev_count))
10 open_name = lib.get_device_name(devenum, 1) ######### 0 or 1 is device number
11 print("\nOpen device " + repr(open_name))
12 device_id = lib.open_device(open_name)
13 print("Device id: " + repr(device_id))
```

Listing 3: Enumeration for found devices and saving the device IDs for later usage

Next the most important functions, which controls the motors, will be explained. To move the motor the function "test_move()" is defined in 4. This function contains four parameters. The first two parameters "lib" and "device_id" are more general than the other ones and will appear on all other explained functions in this section. To point at the library "pyximc", the variable "lib" is passed. The code must also know on which device it should act, so "device_id" is also passed to the function. The other two parameters "distance" and "udistance" are more specific and requires an integer. These tell the motor to move to the position that is at step "distance" and microstep "udistance", regardless of what position the motor was at before. One can also change "command_move()" to "command_movr()", if it is preferred to use the difference between start and end position.

```
1  def test_move(lib, device_id, distance, udistance):
2      print("\textbackslash nGoing to {0} steps, {1} microsteps".format(distance,
       ↪  udistance))
3      result = lib.command_move(device_id, distance, udistance)
4      print("Result: " + repr(result))
```

Listing 4: function which moves device with a certain ID to "distance" and "udistance", where these two represents the steps and microsteps

The next function which is quite important for motor controlling is "test_get_position()". As the name says this function returns the current position of the motor by returning the steps and microsteps.

```
1  def test_get_position(lib, device_id):
2      print("\textbackslash nRead position")
3      x_pos = get_position_t()
4      result = lib.get_position(device_id, byref(x_pos))
5      print("Result: " + repr(result))
6      if result == Result.Ok:
7          print("Position: {0} steps, {1} microsteps".format(x_pos.Position,
           ↪  x_pos.uPosition))
8      return x_pos.Position, x_pos.uPosition
```

Listing 5: function returns current position of device with certain ID

Like the position can be manipulated the speed can be too. Hence there are two more equivalent functions for the motor speed. First the function "test_set_speed()". The parameter "speed" sets the speed in steps per seconds. If one wish to change the speed in microsteps per seconds too, one can add a new parameter "uspeed" and the line "mvst.uSpeed = int(uSpeed)" like line 11 in 6.

```
1  def test_set_speed(lib, device_id, speed):
2      print("\textbackslash nSet speed")
3      # Create move settings structure
4      mvst = move_settings_t()
5      # Get current move settings from controller
6      result = lib.get_move_settings(device_id, byref(mvst))
7      # Print command return status. It will be 0 if all is OK
8      print("Read command result: " + repr(result))
9      print("The speed was equal to {0}. We will change it to {1}".format(mvst.Speed,
       ↪  speed))
10     # Change current speed
11     mvst.Speed = int(speed)
12     # Write new move settings to controller
13     result = lib.set_move_settings(device_id, byref(mvst))
14     # Print command return status. It will be 0 if all is OK
15     print("Write command result: " + repr(result))
```

Listing 6: function sets the current speed of the motor

Then there is the second function "test_get_speed()". It returns the speed of the motor in steps per seconds. The speed in microsteps per seconds can be returned too by adding "mvst.uSpeed" in line 9 in 7 separated by a comma.

```
1  def test_get_speed(lib, device_id)        :
2      print("\nGet speed")
3      # Create move settings structure
4      mvst = move_settings_t()
5      # Get current move settings from controller
6      result = lib.get_move_settings(device_id, byref(mvst))
7      # Print command return status. It will be 0 if all is OK
8      print("Read command result: " + repr(result))
9      return mvst.Speed
```

Listing 7: function returns the current speed of the motor

To guarantee the usage of the proper microstep mode the function "test_set_microstep_mode_256()" is defined (8). It does change the microstep mode by saving and then changing the engine settings (line 4 and 11) so 256 microsteps fit in one step.

```
1  def test_set_microstep_mode_256(lib, device_id):
2      print("\nSet microstep mode to 256")
3      # Create engine settings structure
4      eng = engine_settings_t()
5      # Get current engine settings from controller
6      result = lib.get_engine_settings(device_id, byref(eng))
7      # Print command return status. It will be 0 if all is OK
8      print("Read command result: " + repr(result))
9      # Change MicrostepMode parameter to MICROSTEP_MODE_FRAC_256
10     # (use MICROSTEP_MODE_FRAC_128, MICROSTEP_MODE_FRAC_64 ... for other microstep
       ↪  modes)
11     eng.MicrostepMode = MicrostepMode.MICROSTEP_MODE_FRAC_256
12     # Write new engine settings to controller
13     result = lib.set_engine_settings(device_id, byref(eng))
14     # Print command return status. It will be 0 if all is OK
15     print("Write command result: " + repr(result))
```

Listing 8: function returns the current speed of the motor

To use another mode one can change the last part of line 11 to following:

Modes	Description
MICROSTEP_MODE_FULL	only uses full steps
MICROSTEP_MODE_FRAC_2	2 microsteps fit in one step
MICROSTEP_MODE_FRAC_4	4 microsteps fit in one step
MICROSTEP_MODE_FRAC_8	8 microsteps fit in one step
MICROSTEP_MODE_FRAC_16	16 microsteps fit in one step
MICROSTEP_MODE_FRAC_32	32 microsteps fit in one step
MICROSTEP_MODE_FRAC_64	64 microsteps fit in one step
MICROSTEP_MODE_FRAC_128	128 microsteps fit in one step
MICROSTEP_MODE_FRAC_256	256 microsteps fit in one step

Listing 9: All microstep modes with regarding description; line 11 would then be of following form: "eng.MicrostepMode = MicrostepMode.MICROSTEP_MODE_{...}"

At last the function "test_wait_for_stop()" will be explained. This function checks in a certain interval whether the motor moves. Only if the motor stops the code continues. The interval is given by the parameter "interval" (in milliseconds). This is useful when the VNA must wait for the motor to go to another location.

```
1  def test_wait_for_stop(lib, device_id, interval):
2      print("\nWaiting for stop")
3      result = lib.command_wait_for_stop(device_id, interval)
4      print("Result: " + repr(result))
```

Listing 10: function returns the current speed of the motor

There are many more functions within libximc. A full documentation of the library can be found on https://libximc.xisupport.com/doc-en/ximc_8h.html. After that, all the preparations are done, for finally controlling the motor with the code.

3.2.2 VNA

In the following the part of the code will be explained, where the computer connects to the VNA and where the computer controls the VNA. Then the structure of the VNA commands will be explained and how to find all these.

At first the connection with the VNA must occur. So the socket library must be imported and a socket must be created:

```
import socket

try:
    #create an AF_INET, STREAM socket (TCP)
    instrumentDirectSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error as msg:
    print('Failed to create socket. Error code: ' + str(msg[0]) + ' , Error message :
    ↪  ' + msg[1])
    instrumentDirectSocket.exit();
```

Listing 11: socket library is imported; socket is created as instrumentDirectSocket

To build the connection the IP and the port is necessary. How to get the IP is written in 3.1.2. As an alternative the host name can be used instead of the IP. Most Keysight Technologies use 5025 as the port. If this port does not work, one can check the specific instrument user guide for details.
In the following code the created socket connects to the proper IP:

```
# Alter this host name, or IP address, in the line below to accommodate your specific
↪  instrument
host = '' # Or you could utilize an IP address.

# Alter the socket port number in the line below to accommodate your
# specific instrument socket port. Traditionally, most Keysight Technologies,
# Agilent Technologies, LAN based RF instrumentation socket ports use 5025.
# Refer to your specific instrument User Guide for additional details.
port = 5025

try:
        remote_ip = socket.gethostbyname( host )
except socket.gaierror:
        #could not resolve
        print('Hostname could not be resolved. Exiting')
        instrumentDirectSocket.exit()
print('Ip address of ' + host + ' is ' + remote_ip)

# Given the instrument's computer name or IP address and socket port number now
# connect to the instrument remote server socket connection. At this point we
# are instantiating the instrument as an LAN SOCKET CONNECTION.
instrumentDirectSocket.connect((remote_ip , port))

print('Socket Connected to ' + host + ' on ip ' + remote_ip)
```

Listing 12: created socket connects to VNA with IP or host name and port

Afterwards it is important to understand how the VNA receives commands and how it process them. At first the example down below shows, that one can send a command with the method "sendall()". This method comes from the library socket.

```
instrumentDirectSocket.sendall("SOURce:POWer:LEVel:IMMediate:AMPLitude -20\n")
```

Listing 13: String is sent to VNA; it commands the VNA to change the power level to $-20dBm$

This string tells the VNA to set the output power level to $-20dBm$. So first the VNA needs the command and afterwards the belonging parameter. The VNA has many functions, so there are subcommands too. These are seperated by a colon. As an example the command "POWer" is a subcommand of the command "SOURce" (13). Therefore it is not enough to just tell the VNA the subcommand you want to execute, but also all commands, which are superior to the subcommand. Even those commands can have parameters. They must be also written after the command, but before the colon.

But there is a another issue. The VNA can not handle the string type, so it is necessary to convert those into bytes. In the next listing (14) the proper way to send a command will be introduced:

```
1 powerLevel = -20     # in dBm
2
3 instrumentDirectSocket.sendall(b"SOURce:POWer:LEVel:IMMediate:AMPLitude " +
  ↪  bytes(str(powerLevel),encoding='utf8') + b"\n")
```

Listing 14: Byte is sent to VNA; it commands the VNA to change the power level to $-20dBm$

To change a string into a byte via python, with standard encoding (utf8), the letter "b" can be put before the string. But if a parameter is requested, one have to use the "bytes()" method. The "bytes()" method needs two parameters. First the string, which will be converted and second the encoding, which can be set to "utf8". In this example the parameter is an integer so the "str()" method can convert it to a string and afterwards it can be converted into a byte. The bytes can be added up with a plus sign.

In the following chart some commands are listed, that are used for the experiment in 4.3.

Commands	Description
b"SENS:FREQ:CENTer " + centerFrequency + b";SPAN " + frequencySpan + b"\n"	sets frequency center and span
b"DISPlay:WINDow1:TRACe1:Y:SCALe:AUTO;*OPC?\n"	autoscales y axis
b"SENse:SWEep:MODE CONT;*OPC?\n"	sets trigger to continuous
b"SENse:SWEep:MODE SINGle;*OPC?\n"	sets trigger to single/hold
b"SOURce:POWer:LEVel:IMMediate:AMPLitude " + powerLevel + b"\n"	sets power level
b"CALCulate:MEASure:FORMat MLOGarithmic\n"	sets format to logarithmic
b"CALCulate:MEASure:FORMat MLINear\n"	sets format to linear
b"CALCulate:MEASure:FORMat GDELay\n"	sets format to delay
b"SENSe1:SWEep:POINts " + sweepPoints + b"\n"	sets number of sweep points
b"SENSe1:BANDwidth:RESolution " + ifBandWidth + b"\n"	sets IF bandwidth
b"SENSe:AVERage:STATe ON(OFF)\n"	turn average mode ON(OFF)
b"SENSe:AVERage:COUNt integer\n"	sets number of averaging points
b"MMEMory:STORe '" + fileURL + b"'\n"	saves s2p file for s-parameters
b"SENSe:CORRection:CSET:ACTivate " + calName + b",1\n"	turns measured calibration file on

Listing 15: List of VNA commands with description; variables are in red and should be from type bytes like explained above ("integer" is an exception and should be an integer)

These are just a couple of all the commands. To find any command, one can use the so-called "Command Finder" ([On4]). It can be found on the VNA itself, by go in the "NAHelp" menu on the top side of the screen. A window will appear, where some options are listed. Under "Programming" one can find a short explanation and a link called "Command Finder". After clicking the link the menu down below (4) appears on the screen.

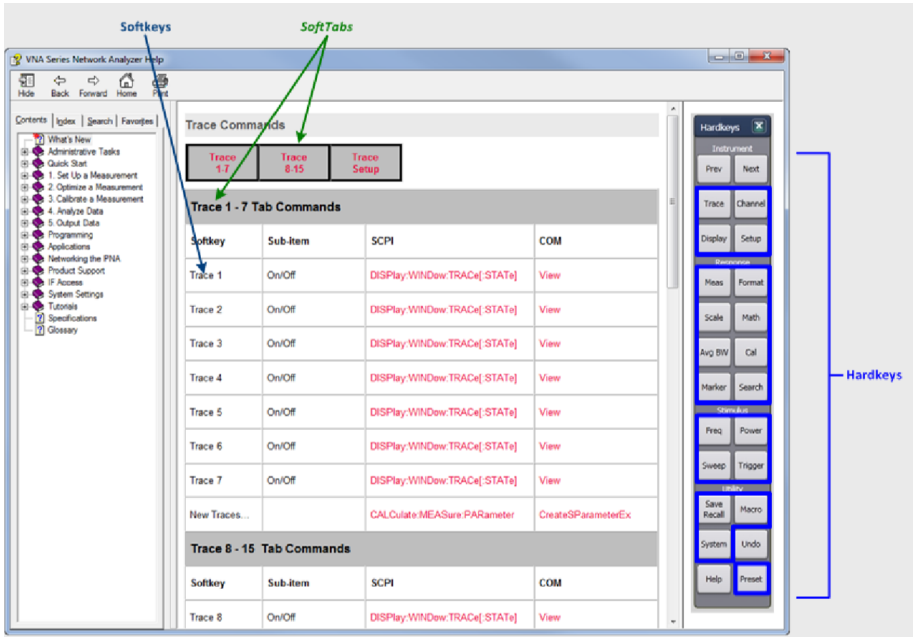

Figure 4: "Command Finder" menu for looking up commands to communicate with the VNA

This menu can be used to navigate to the desired commands. On the right side there are the hardkeys, which represents all physical buttons on the VNA. In the middle the softkeys are listed. The softkeys represent the digital buttons on the display of the VNA. The whole explanation and documentation of the command will show up, by clicking the links right next to the regarding softkey. One can choose between SCPI for a wireless connection and COM for a serial connection.

Finally, how the code selects specific calibration sets is explained. The associated command has already been introduced in the table above. However, it may be unclear which parameter to pass to this command. To find out, one can navigate to the selection menu of the calibration sets on the VNA. Then there is a list of all VNA calibration measurements that are saved (5).

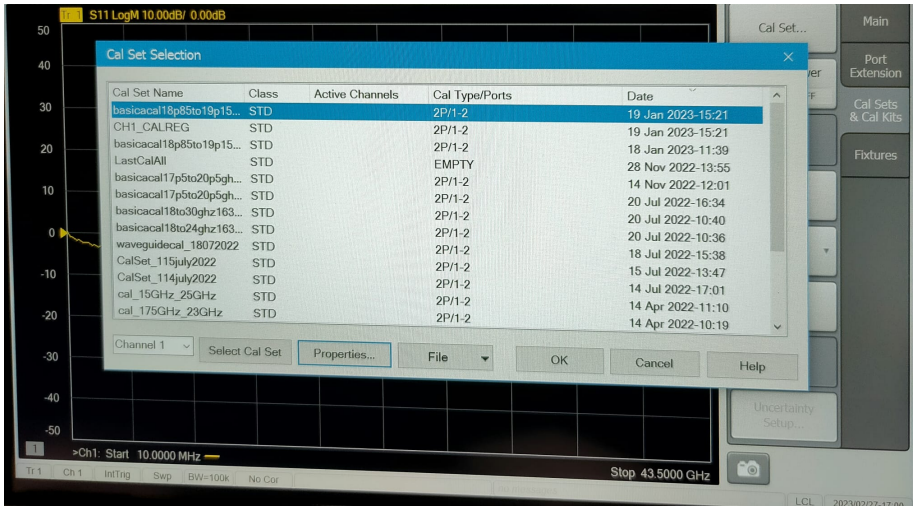

Figure 5: Calibration set selection; in this menu all saved calibration measurements are listed

If one now selects the desired set and clicks on properties, a new window opens (6). There is some information in this window. The now relevant information is the so-called "GUID". This is a string and specifies the calibration measurement.

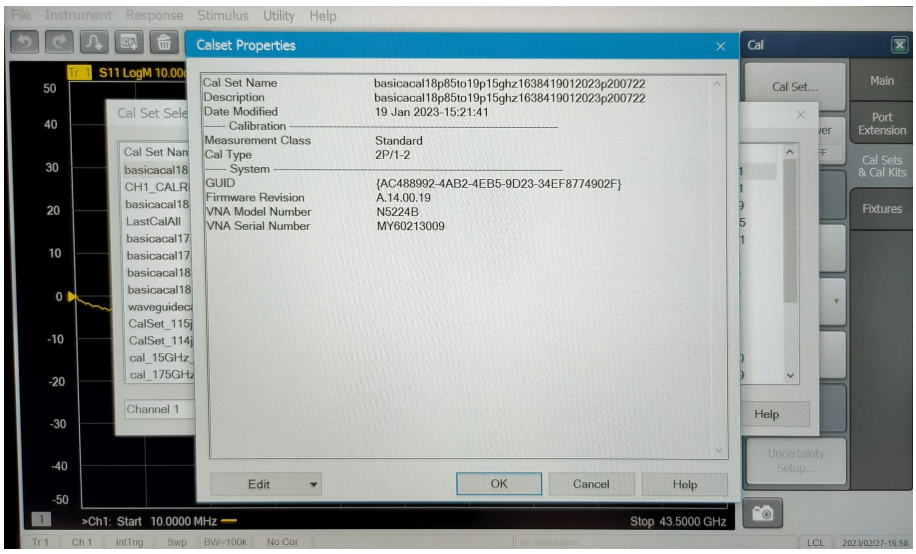

Figure 6: Calibration set selection; in this menu all saved calibration measurements are listed

It is then sufficient to specify this in the VNA command to guarantee the selection of the set. The proper execution within the code would then be following:

```
calName = bytes('"{AC488992-4AB2-4EB5-9D23-34EF8774902F}"',encoding='utf8')

instrumentDirectSocket.sendall(b"SENSe:CORRection:CSET:ACTivate " + calName +
↪  b",1\n")
```

Listing 16: "GUID" is encoded to bytes type; code can communicate with VNA, which calibration set is chosen

3.2.3 Example code for measurement

In this chapter the parts of the code doing the measurement and the preset are explained. This serves as an example to illustrate the possibilities of a code. In 4 the regarding application example is explained. There are several functions that act on the devices. Two of them will be presented.

The function "measurement()" (17) sets the microstep mode to 256 and the motor speed to 250 steps per seconds. Then the motor moves to step zero and microstep zero. The code waits till the motor gets there and then the measurement loop starts. It moves to a position, waits till it stops and then saving the measured s2p file. Then the loop repeats.
The benefit of such loop is to do huge measurements with small effort.

```
1  def saveS2P(fileURL):
2      instrumentDirectSocket.sendall(b"MMEMory:STORe '" +
       ↪  bytes(fileURL,encoding='utf8') + b"'\n")
3      return True
4
5  def measurement():
6      test_info(lib, device_id)
7      test_status(lib, device_id)
8      test_set_microstep_mode_256(lib, device_id)
9      test_set_speed(lib, device_id, 250)
10     test_move(lib, device_id, 0, 0)
11     test_wait_for_stop(lib, device_id, 100)
12     startpos, ustartpos = test_get_position(lib, device_id)
13
14     for i in range(17):
15         test_move(lib, device_id, -1000*i, 0)
16         test_wait_for_stop(lib, device_id, 100)
17         saveS2P("d:/Nils/meas_" + str(1000*i) + ".s2p")
18         time.sleep(3)
19
20     test_move(lib, device_id, 0, 0)
21     test_wait_for_stop(lib, device_id, 100)
22     return True
```

Listing 17: Measurement instructions for VNA and motor

Afterwards a VNA setup function called "instrumentSimplifiedSetup()" (18) will be explained. It starts with performing a factory preset (line 4). This can reduce problems and prevent unwanted settings. Afterwards some general settings are done. In line 8 the first window on the VNA is turned on and in line 12 a measurement called "MyMeas" is defined. Within the same command the measurement parameter, which represents one of the four scattering parameters, is assigned to the defined measurement itself. This can be important if one does different measurements on different scattering parameters. In line 15 the previously defined measurement is associated with the first trace on the visible window. Next some more specific settings are done. In line 18 a certain calibration called "calName" is set. This command can allow dynamic switching between calibrations. Then some values like the power level (line 21) or the frequency range (line 28) are set. Also some additional commands are used like turning off the averaging (line 24) or set the format to logarithmic (line 37). This shows the flexibility of such code and the possibility of creating an individual automatically setup.

```
def instrumentSimplifiedSetup():
    try :
        # Perform a factory preset with removal of all traces, windows, etc.
        instrumentDirectSocket.sendall(b"SYSTem:FPReset;*OPC?\n")
        opComplete = instrumentDirectSocket.recv(8)

        # Turn on Window 1
        instrumentDirectSocket.sendall(b"DISPlay:WINDow1:STATE ON\n")

        # Define a measurement name 'MyMeas' (a.k.a. label),
        # and  a measurement parameter (e.g. S21, S11, or other)
        instrumentDirectSocket.sendall(b"CALCulate:PARameter:DEFine:EXT
        ↪  'MyMeas',S11\n")

        #Associate ("FEED") the measurement name ('MyMeas') to WINDow (1), and give
        ↪  the new TRACe a number (1).
        instrumentDirectSocket.sendall(b"DISPlay:WINDow1:TRACe1:FEED 'MyMeas'\n")

        # Set calibration
        instrumentDirectSocket.sendall(b"SENSe:CORRection:CSET:ACTivate " +
        ↪  bytes(calName,encoding='utf8') + b",1\n")

        # Set power level
        instrumentDirectSocket.sendall(b"SOURce:POWer:LEVel:IMMediate:AMPLitude " +
        ↪  bytes(str(powerLevel),encoding='utf8') + b"\n")

        # Set average ON or OFF
        instrumentDirectSocket.sendall(b"SENSe:AVERage:STATe OFF\n")
        #instrumentDirectSocket.sendall(b"SENSe:AVERage:COUNt 50\n")

        # Set center and span frequencies.
        instrumentDirectSocket.sendall(b"SENS:FREQ:CENTer "
        ↪  +bytes(str(centerFrequency),encoding='utf8') +b";SPAN "
        ↪  +bytes(str(frequencySpan),encoding='utf8') +b"\n")

        # Set number of sweep points
        instrumentDirectSocket.sendall(b"SENSe1:SWEep:POINts
        ↪  "+bytes(str(sweepPoints),encoding='utf8')  +b"\n")

        #Set the bandwidth of the digital IF filter to be used in the measurement.
        instrumentDirectSocket.sendall(b"SENSe1:BANDwidth:RESolution
        ↪  "+bytes(str(ifBandWidth),encoding='utf8') +b"\n")

        # Set format to log
        instrumentDirectSocket.sendall(b"CALCulate:MEASure:FORMat MLOGarithmic\n")

        # PNA requires all CALCulate sub commands to operate on a prior selected
        ↪  measurement name.
        instrumentDirectSocket.sendall(b"CALCulate:PARameter:SELect
        ↪  'MyMeas';*OPC?\n")
        opComplete = instrumentDirectSocket.recv(8)

    except socket.error:
        #Send failed
        print('Send failed')
        instrumentDirectSocket.exit()
```

Listing 18: Example of a simple VNA setup function

4 Application example

To demonstrate the before discussed automation (3) the so-called Bead pull method will be presented. In 2.2 this methods theory is introduced.

4.1 Setup

First of all there is the VNA and the stepper motor, which are mentioned in 3.1. Then there is a copper plate, which from now on will be called mirror, because it is necessary to reflect the microwaves almost completely. The microwaves are produced by the VNA and will be sent by an antenna. This antenna sends the microwaves as linear polarized spherical waves. Therefore in this setup a parabolic mirror will be installed to focus the signal and get rid of the losses. As a perturbation a so-called bead will be pulled by a string (Bead pull method). To build the bead aluminium foil is rolled up. For the string dental floss is used.

The VNA is connected to the antenna via port 1 and sends the produced waves through the antenna. It is seen that only port 1 is used and so only S_{11} will be important to make statements about how the wave is perturbed (2.1). The antenna is pointed to the parabolic mirror, which will reflect the signal in a 90 degree angle towards the non-parabolic mirror. That mirror reflects the waves in a 180 degree angle, so the signal do the same way twice. Between those two mirrors the string, three rods and the motor are installed. The bead is attached to the string. Two of the rods both with a big enough hole on the top are used to keep the height of the string on the same level and are therefore immobile. The string is passed through the holes. To guarantee the tension of the string it is fixed with a weight on the one side and with another rod onto the stepper motor on the other side. So the location of the motor can describe the location of the bead. Because the motor moves in steps and microsteps it must be calibrated.

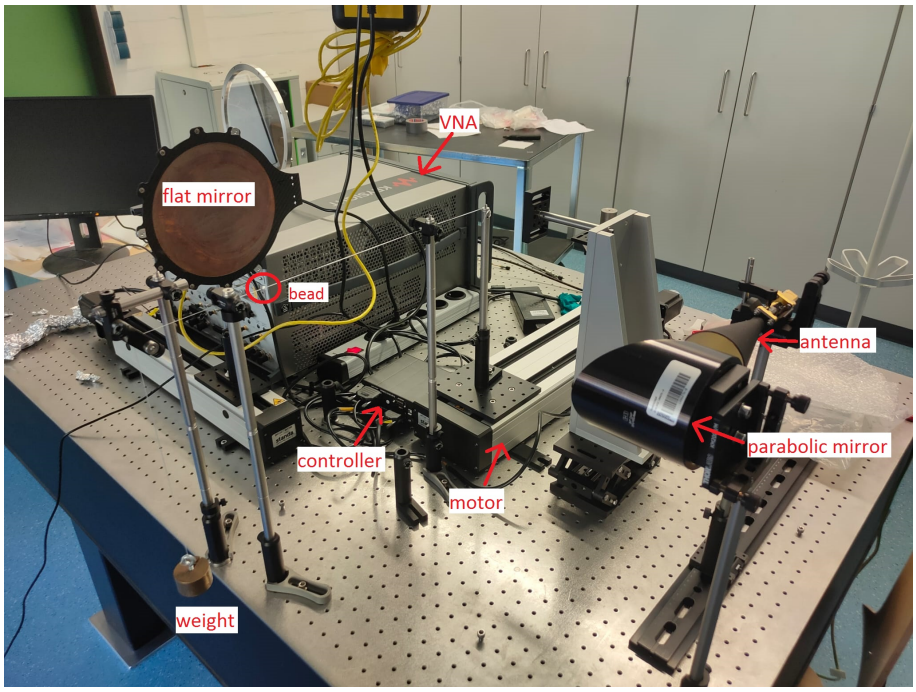

Figure 7: setup with all important components labeled

4.2 Calibration of motor

To figure out how many microsteps fit in one step one must know that there are different microstep-modes. To maximize the sensitivity of the motor the use of the microstep-mode "256" is recommended, because in this mode as the name says 256 microsteps fit in one step. This setting is more explained in 3.2.1 and within how to change the mode as well. To cover the entire area of the motor 22 equidistant spots are measured with a ruler from one fix point. The following chart shows the measured values:

steps [10^3]	0	1	2	3	4	5	6	7	8	9	10
length [mm]	36	49	61	74	86	98	112	124	136	149	161
steps [10^3]	11	12	13	14	15	16	17	18	19	20	21
length [mm]	174	187	199	211	224	236	249	261	274	286	299

Listing 19: values of calibration measurement

Because the length grows linear with the steps, one can do a linear fit through the measured points to evaluate the conversion factor. Hence the length l has the following relationship with the steps s, where δ is the slope and r is the offset:

$$l = \delta \cdot s + r \tag{10}$$

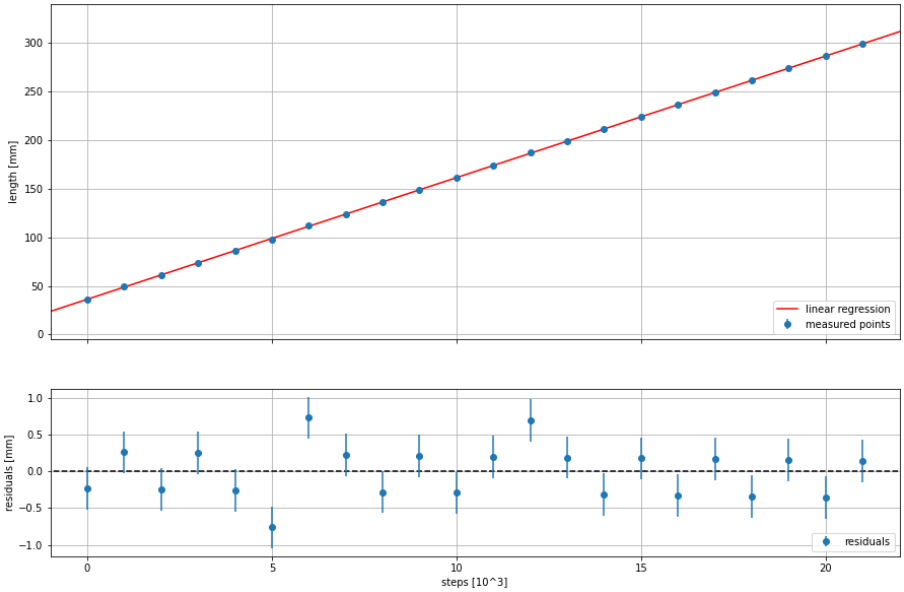

Figure 8: on top linear fit, on bottom residuals; $\frac{\chi^2}{n_{dof}} = 1.679$

As one can see in 8 the conversion factor of steps into length is the slope of the linear fit. It is:

$$\delta = (12.506 \pm 0.013)\frac{mm}{10^3} = (12.506 \pm 0.013)\mu m \tag{11}$$

The error comes from the assumption, that a statistical error of $\frac{1}{\sqrt{12}}mm$ on the length occurred during the measurement, because a normal ruler was used to measure the length. A normal ruler has an resolution of $1mm$ and the uncertainty is normally distributed, hence a factor of $\frac{1}{\sqrt{12}}$ is multiplied by $1mm$. The error on δ comes then from the regression itself. The quality of the measurement is described by $\frac{\chi^2}{n_{dof}} = 1.679$. Although this value does not speak for a perfect measurement, no special systematics can be determined from the residuals. This is also reflected in the comparison of the measured δ and the resolution of $12.5\mu m$ specified by the manufacturer (https://www.standa.lt/products/catalog/motorised_positioners?item=305). However, this is acceptable for the purpose of this demonstration.

4.3 Demonstrative measurement

For the demonstrative measurement the bead is positioned at the edge of the one side of the mirror. The VNA then measures the complex perturbed reflection coefficient $S_{11,p}$ in a certain frequency range. After this the motor moves the bead to another position and the whole process is repeated (Chapter 3.2.3). For the unperturbed case the first measurement points (at step zero and microstep zero) are used to describe the complex unperturbed reflection coefficient $S_{11,u}$, because when the bead is at the edge the focused microwaves does not interact with it. The following measurement parameters were used:

frequency lower bound	$17.5GHz$
frequency upper bound	$20.5GHz$
IF bandwidth	$50kHz$
sweep points	2^{14}
power level	$-20dBm$

Listing 20: Measurement parameters

Then with equation 9 a relation between those measured reflection coefficients, the frequency and the absolute value of the electric field can be displayed in a two dimensional heatmap. The y axis represents the frequency, the x axis the position of the bead and the color the absolute value of the electric field. By setting the offset to zero (equation 10) one can use the conversion factor δ to transform the steps into length. Because the absolute value of the electric field depends on a geometric factor k_{ST} the electric field is shown with arbitrary units. However, the image remains independent of k_{ST}.

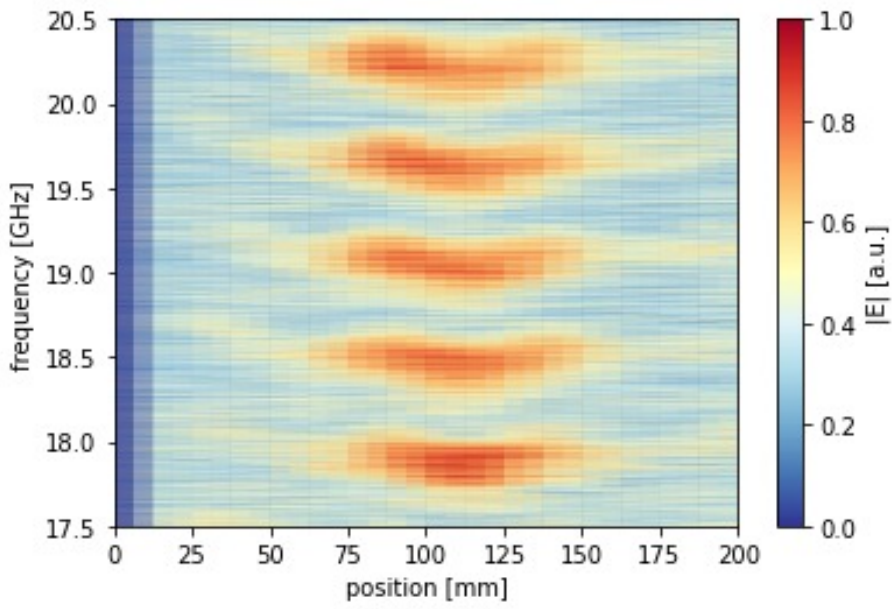

Figure 9: heatmap with position and frequency as x and y axis; color is normalized absolute value of electric field; plot is linearly interpolated between the grid points (not interpolated plot can be found in appendix)

5 Conclusion

To conclude the thesis it can be said that there are certainly some things that can still be improved or added. For example, to control the motors wirelessly or to send the saved measurement data back to the computer. However, this should not diminish the success of this work. After the measurement was successful, it can be said in retrospect that the work was a complete success. The automation, both with regard to the VNA and that of the motors, works flawlessly.

6 Appendix

6.1 Graphics

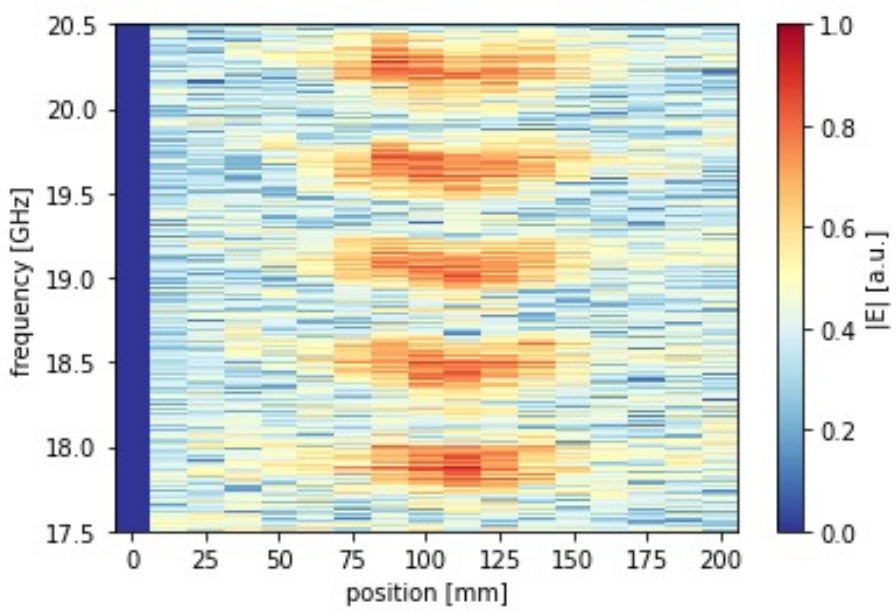

Figure 10: not interpolated heatmap with position and frequency as x and y axis; color is normalized absolute value of electric field

References

[Ste66] Charles W Steele. "A nonresonant perturbation theory". In: *IEEE Transactions on Microwave theory and Techniques* 14.2 (1966), pp. 70–74.

[PQ77a] Roberto D Peccei and Helen R Quinn. "Constraints imposed by CP conservation in the presence of pseudoparticles". In: *Physical Review D* 16.6 (1977), p. 1791.

[PQ77b] Roberto D Peccei and Helen R Quinn. "CP conservation in the presence of pseudoparticles". In: *Physical Review Letters* 38.25 (1977), p. 1440.

[Wu97] Dan-di Wu. "A brief introduction to the strong CP problem". In: *Zeitschrift für Naturforschung A* 52.1-2 (1997), pp. 179–182.

[CS00] Edvige Corbelli and Paolo Salucci. "The extended rotation curve and the dark matter halo of M33". In: *Monthly Notices of the Royal Astronomical Society* 311.2 (2000), pp. 441–447.

[Bak+06] CA Baker et al. "Improved experimental limit on the electric dipole moment of the neutron". In: *Physical Review Letters* 97.13 (2006), p. 131801.

[Mos+09] A Mostacci et al. "About non resonant perturbation field measurement in standing wave cavities". In: *Proceedings of PAC09* (2009), pp. 3407–3409.

[Cal+19] A Caldwell et al. "A new experimental approach to probe QCD axion dark matter in the mass range above 40 μeV". In: *The European physical journal/C* 79.PUBDB-2019-01463 (2019), p. 186.

[On1] https://edadocs.software.keysight.com/kkbopen/a-python-programming-example-utilizing-direct-socket-connection-for-lan-based-instrument-control-outside-of-and-independent-of-the-use-of-pyvisa-577935557.html

[On2] http://files.xisupport.com/Software.en.html

[On3] https://github.com/NilsDeimann/Automation-of-a-vector-network-analyzer-and-a-stepper-motor

[On4] https://www.keysight.com/de/de/lib/resources/help-files/n52xxb-pna-series-help-file.html