MIDDLE EAST TECHNICAL UNIVERSITY
ELECTRICAL – ELECTRONICS ENGINEERING DEPARTMENT
EE400
SUMMER PRACTICE REPORT

Student Name: Alkım Bozkurt

Student ID: 2515674

Summer Practice Place Name: Vienna University of Technology (TU Wien)

Division: Institute of Electrodynamics, Microwave and Circuit Engineering

Summer Practice Dates: 24.06.2024 to 26.08.2024

Submission Date: 14.10.2024

Supervisor: Assoc. Prof. DI Dr. Holger Arthaber

Contact Information of the Supervisor:

Email: holger.arthaber@tuwien.ac.at

Phone Number: +43 1 58801 – 35420

TABLE OF CONTENTS

1.INTRODUCTION

I have conducted my mandatory internship at Vienna University of Technology or as known as TU Wien. TU Wien can be considered as the largest scientific research institution among Austria. I wanted to experience working with in an academical environment and therefore I tried to secure an internship position at the university. I have contacted with principal investigator and made interviews prior to my acceptance to my internship.

As a student starting to my senior year, I wanted to specialize in the field of RF and Microwave Engineering. Therefore, I applied for an internship at the *Institute of Electrodynamics, Microwave and Circuit Engineering (EMCE)*, which can be considered as one of the most important research institutions among TU Wien. The institute currently runs many research projects related to fields microwave and high-frequency circuits. There were three main research branches among the Institute, and I had the pleasure to be a short-term participant of the Microwave Engineering group.

During my internship, I was supervised by the principal investigator of our research group Assoc. Prof. DI Dr. Holger Arthaber. Mr. Arthaber is an experienced engineer and researcher, and he manages remarkable projects with significant amount of fundings. Throughout my internship, his guidance was essential for me to finish my project and learn the fundamental concepts. In addition, I was able to learn new techniques and develop my skills during my internship thanks to his approach. There were also PhD and master's students in our research group, and they also helped me a lot during my internship.

My task was replacing an existing RF impedance tuner controller with a new one. I was expected to increase the functionality of the old RF impedance tuner controller and meet the specifications given by my supervisor while developing the new tuner controller. I have started my internship by analysing and understanding the behaviour of the old tuner controller. I have done some measurements on impedance tuners using the old controller in order to understand the characteristics of the old tuner controller. Then, I tried to develop a new controller by using a motor controller board and the observations that I have done before. I have developed a MATLAB class which was the main software of the new RF tuner controller. After crafting the new RF tuner controller in terms of hardware and software, I have done some automated tests on the new RF tuner controller in terms of RF characteristics. With these measurements and test I have tried to tune and optimize the behaviour of the new controller by using RF measurement tools. At the end of the project, I was able to build a new tuner controller with reliable and reproducible RF characteristics.

Throughout the project I was given to complete very different tasks such as soldering cables to developing a MATLAB class. Thanks to this, I have learned some crucial skills and methods used in RF field and I gained a remarkable understanding of RF concepts.

In this report I briefly describe the university and institute. Then through the following chapters I will try to explain and elaborate on the work that I have done during my internship. I will use graphics, figures and images to emphasize the crucial points in a better way.

2. DESCRIPTION OF THE INSTITUTION

2.1 Summer Practice Place Name

Vienna University of Technology (Technische Universität Wien – TU Wien)

2.2 Contact

Address: Technische Universität Wien, Institute of Electrodynamics, Microwave and Circuit Engineering, Gußhausstraße 25/354, 1040 Wien, Austria,

Phone: +43 1 58801 354 02

E-mail: josef.wieser@tuwien.ac.at

2.2 Review

The Vienna University of Technology (TU Vienna) stands as Austria's largest institution for research and education in science and engineering. The university accommodates more than 42,000 students, boasts total assets nearing 460 million euros, and employs around 8,800 individuals. There are eight faculties and numerous institutions which are part of TU Wien. The buildings of these institutions are all distributed along the city of Vienna. The university has hosted numerous Nobel Prize winners. The latest one is Ferenc Krausz who has wone the Nobel prize in the field of physics in 2023. Latest statistics are as of April 2021: 43,800 students, 5,300 graduates per year and 10,200 employees.

Figure 1 – Faculty of Electrical Engineering and Information Technology

Figure 2 – Institute of Electrodynamics, Microwave and Circuit Engineering

Institute of Electrodynamics, Microwave and Circuit Engineering is a research center under the Faculty of Electrical Engineering and Information Technology. The institute conducts both basic and application-oriented research in the calculation of electromagnetic fields, microwave technology for information and communication technology, the development and implementation of analog, mixed-signal, and opto-electronic circuits, as well as THz sources, detectors, and systems. The team consists of experienced professors, PhD and master's students, lecturers and technicians. The institute also hosts graduate programs in the fields of telecommunication, microelectronics, automation technology, etc. There are three main research units connected to the institute. These are:

1) The Microwave Engineering Group
2) Research unit of THz Electronics
3) Research unit of Integrated Circuits

The Microwave Engineering Group engages in research ranging from fundamental to applied studies, leveraging a state-of-the-art microwave lab and advanced simulation tools for circuits, systems, and electromagnetic fields. Their expertise extends beyond microwave technology, encompassing FPGA-based real-time processing and intricate system design. The group primarily focuses on communication applications, developing efficient, linear transceivers and resilient receivers. They also excel in UHF RFID reader design and tag tracking, while their latest research explores interference modelling in shared frequency bands.

2.3 Brief History

The Technical University was founded in 1815 as the Imperial and Royal Polytechnic Institute by Emperor Franz I of Austria to train engineers for the military, mining, and civil sectors. The institute opened on November 6, 1815, with 47 students and three professors. Construction of the Karlsplatz building began in 1816, and the institute moved there in 1818. In 1872, it became the Technical University, gaining the right to award doctorates in 1901.

3. RF TUNERS AND TUNER CONTROLLERS

3.1 RF IMPEDANCE TUNERS

RF Tuners or to define more precisely, RF impedance tuners are RF devices which are able to provide impedances with arbitrary magnitudes and phases. For RF applications, mostly during testing and developing phases, arbitrary non-50 Ω impedances are required. In order to provide this, RF impedance tuners are utilized. Usually there are slabs and stepper motors inside the impedance tuner. The positions of reflecting plates are adjusted with the help of these motors. By being able to control the reflecting plates, the one can control the properties of the reflecting electromagnetics signal. Also, thanks to this RF impedance tuners have controllable impedance. There are two main types of impedance tuners, namely manual and automated tuners. Nowadays, this controlling process is done mostly in an automated way. The one that I have used during my project was called an automated slide-screw impedance tuner. The model's name was Maury Microwave Automated RF Impedance Tuner MT982A02. It had three stepper motors inside which were able to slide inside and insert slabs which were interrupting the electrical signal incident from one port to the other. This interruption was giving us the access to change the phase and magnitude value of the impedance

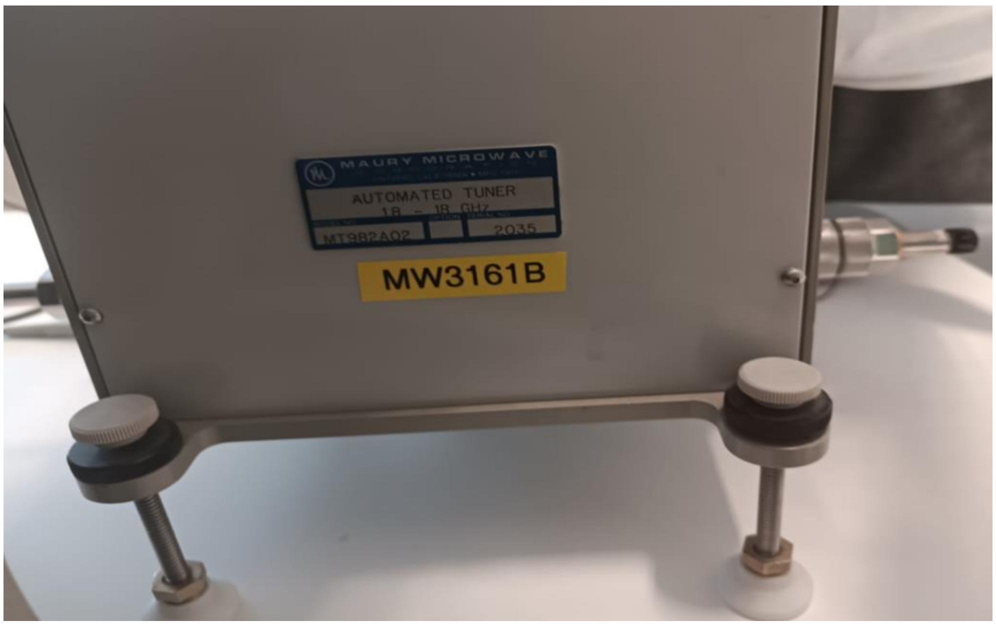


Figure 3 – RF Impedance Tuner Maury Microwave MT982A02

3.2 RF IMPEDANCE TUNER CONTROLLERS

As I have mentioned in the previous chapter, the movement of slabs thanks to the movement of motors is the main phenomenon which enables us to control the impedance. In other words, control of impedance is directly related to the control of stepper motors inside the RF tuner. Therefore, a need for a tuner controller arises. These controllers simply execute movement commands to motors by controlling the phase current. There are also some other advanced functionalities of the tuner controllers, but the most important reason to use them are to adjust the positions of the motors. Most of these tuner controllers are operated by the end-user. The RF impedance tuner controller that I was given to replace was an old controller from 2004. It was a Maury Microwave product. The main problem was motors were being controlled one-by-one, which was slowing down the whole process. One of the key requirements for the new controller was to move the stepper motors inside the RF impedance tuner controller simultaneously.

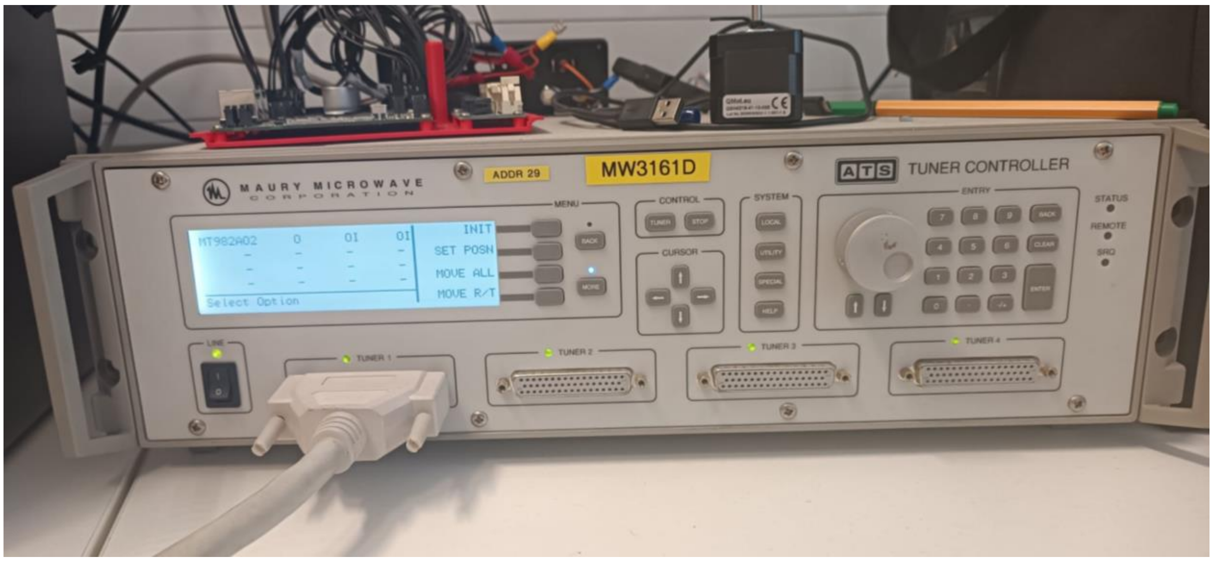


Figure 4 – RF Impedance Tuner Controller Maury Microwave MT986C

4. CHARACTERIZATION OF MOTOR CONTROL

In order to build a new tuner controller, it was a must to understand the behaviour and logic behind the old tuner controller. In order to achieve this some experiments and measurements were conducted. There were three stepper motors inside the RF tuner which all had different purposes, namely Motor 1, Motor 2 and Motor 3. To understand the movements of the mentioned motors reliably, the phase currents were observed.

4.1 Measurement of Phase Currents

To be able to have a mathematical, reliable, logical and objective understanding of the movement of the motors, the phase currents of the motors were observed. To perform the measurements, current sensors and an oscilloscope were utilized. Since these measurements were not done once or twice, an automated setup for measurement was a need. This was achieved by using the VISA interface and controlling the measurement instruments via a MATLAB script.

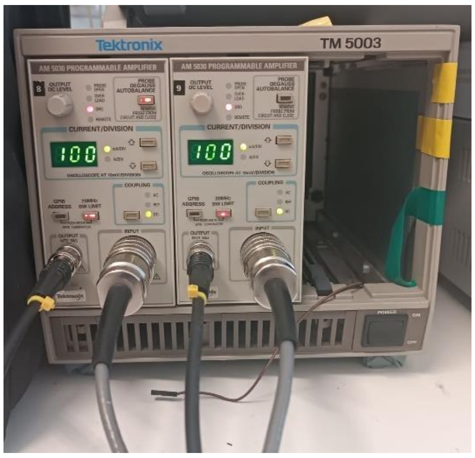

Tektronix
TM 5003
AM 5030 PROGRAMMABLE AMPLIFIER
AM 5030 PROGRAMMABLE AMPLIFIER
100
100

Figure 5 – Current sensor

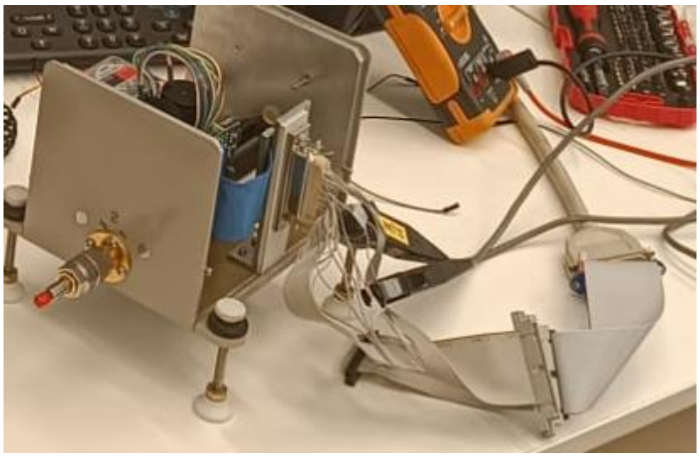


*Figure 6 - P*robes connected to motor phase cables

The phase currents were observed from the oscilloscope and then the waveform data was saved to MATLAB. The dataset of phase currents was then utilized for obtaining more advanced information about motor movements, which will be discussed in the next chapter.

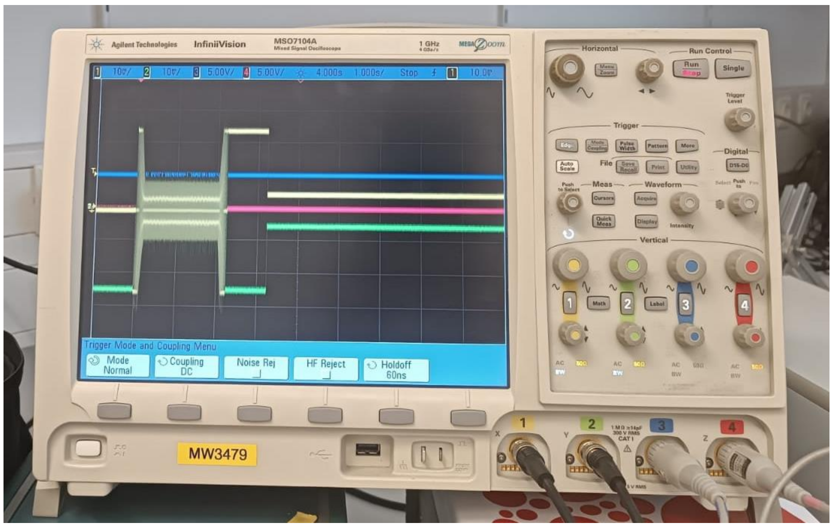


Figure 7 – Current waveform on DSO

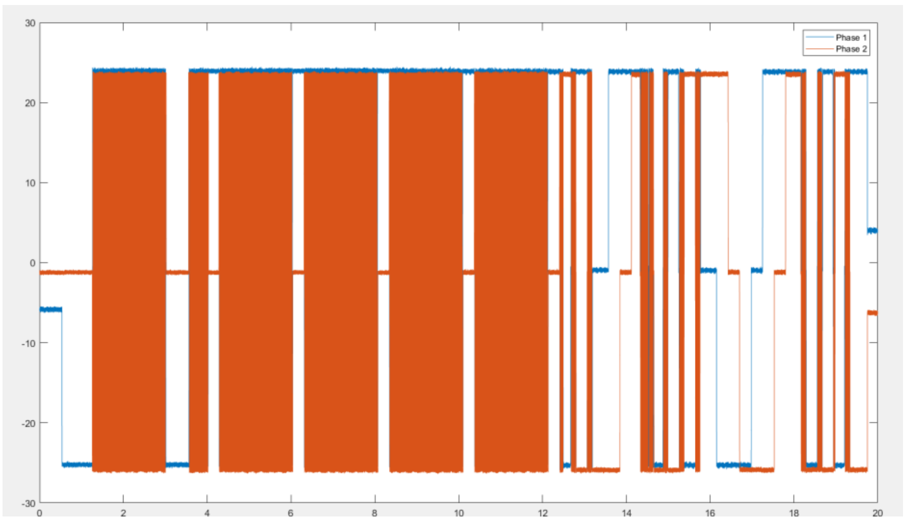
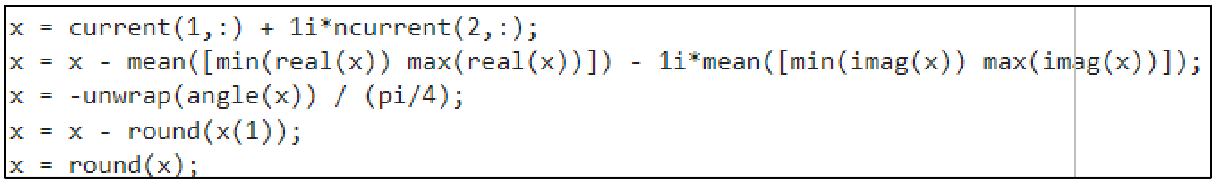


Figure 8 – Motor phase current waveforms during a movement

In the Figure 8, the one can see that the movement is controlled by adjusting the phase current. However, the maximum current, also known as the "run current" value is always the same. The movement is controlled by adjusting the intervals for the run current rather than changing the current magnitude.

4.2 Obtaining Position Characteristics from Phase Currents

The current waveform was not sufficient to understand the movement behaviour. Therefore, some other properties were deducted. The main properties for a movement can be easily considered as current velocity and current position. Those properties were crucial and should had been obtained.



```
x = current(1,:) + 1i*ncurrent(2,:);
x = x - mean([min(real(x)) max(real(x))]) - 1i*mean([min(imag(x)) max(imag(x))]);
x = -unwrap(angle(x)) / (pi/4);
x = x - round(x(1));
x = round(x);
```

Figure 9 – Obtaining position vs. time from current vs. time

The stepper motor inside the impedance tuner is a bipolar stepper DC motor. In other words, there are two phases which are responsible from the movement of motor. By adjusting the interval as seen in Figure 7 and Figure 8, the rotor is span with arbitrary velocity. It is a known fact that these two-phase currents are out of phase. Therefore, the information of two-phase currents can be stored as a single complex number. This was done in the first code line in Figure 9. Then, to assign the starting position as position 0, the mean was subtracted from this complex number. The phase difference between two phase currents is always 90° throughout the movement. However, as the magnitude of phase current changes, the phase angle for the complex number also changes.

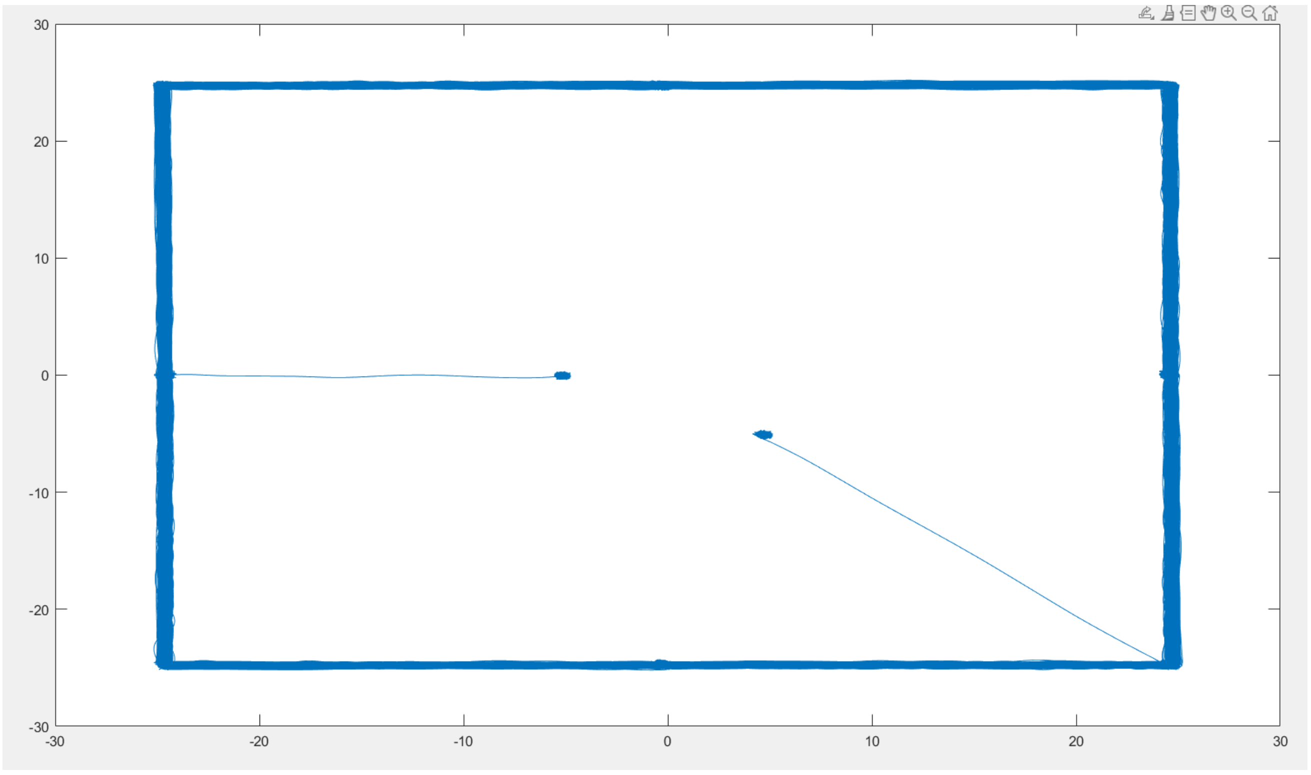


Figure 10 – Complex number obtained from current throughout the movement on complex plane

The rotation of complex number on the complex plane in Figure 10 directly represents the rotation of the motor. The two points closer to the center indicates the starting and the stopping position of the motor. By obtaining the phase angle of the complex number I was able to track the position of the movement as it rotates.

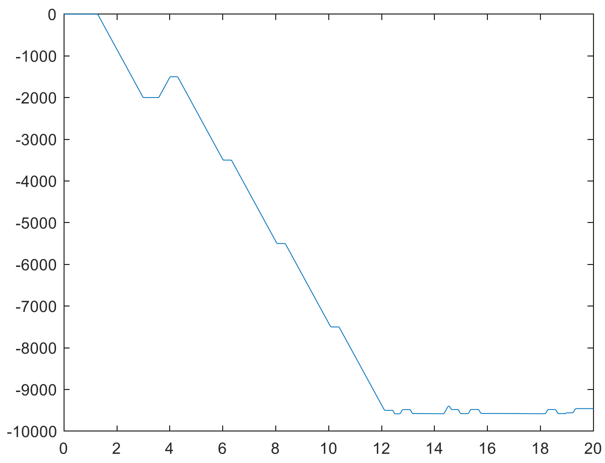
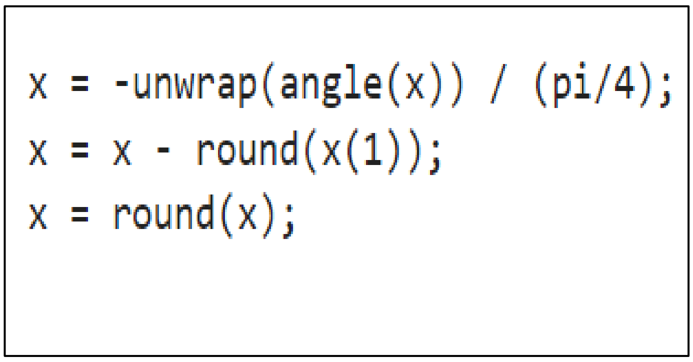


Figure 11 – Position versus time obtained



```
x = -unwrap(angle(x)) / (pi/4);
x = x - round(x(1));
x = round(x);
```

Figure 12 – Obtaining position from angle

After obtaining the rotating phase angle, the angle travelled was known. To convert this information into a reliable position information I had to divide this phase angle value to 360° as a full rotation takes that much angle to travel. This value represents the full rotations done versus time. To convert this information into current position versus time, the micro-stepping configuration of the motor was taken into account. In other words, how much step it takes to complete a full rotation was considered. This can be seen in Figure 12. Finally, the information of current position vs. time was obtained.

4.3 Limit Switches and Switch Signals

Tuner controller was able to detect whether a motor hits the limit or not from the light barrier sensors which were installed inside the RF impedance tuner. These sensors simply provide logic signals to tuner controller and tuner controller utilizes these signals to regulate the movement of motors safely. This signal was also utilized significantly in the new controller. When a motor hits a lower or higher limit, the switch signals become low, and this indicates it is an active-low signal.

4.4 Obtaining Velocity Characteristics

After being able to access the position information, the last crucial property of a movement, velocity was the only key parameter to obtain. This was done by simple differentiation and smoothing processes applied to position dataset. By obtaining these important properties and matching them with the logic signal coming from switch sensors, the electrical characterization of control of movement of a motor was completed. After combining this information, any pre-defined sequence by the old controller could be imitated by the new tuner controller. The one can see the figure below.

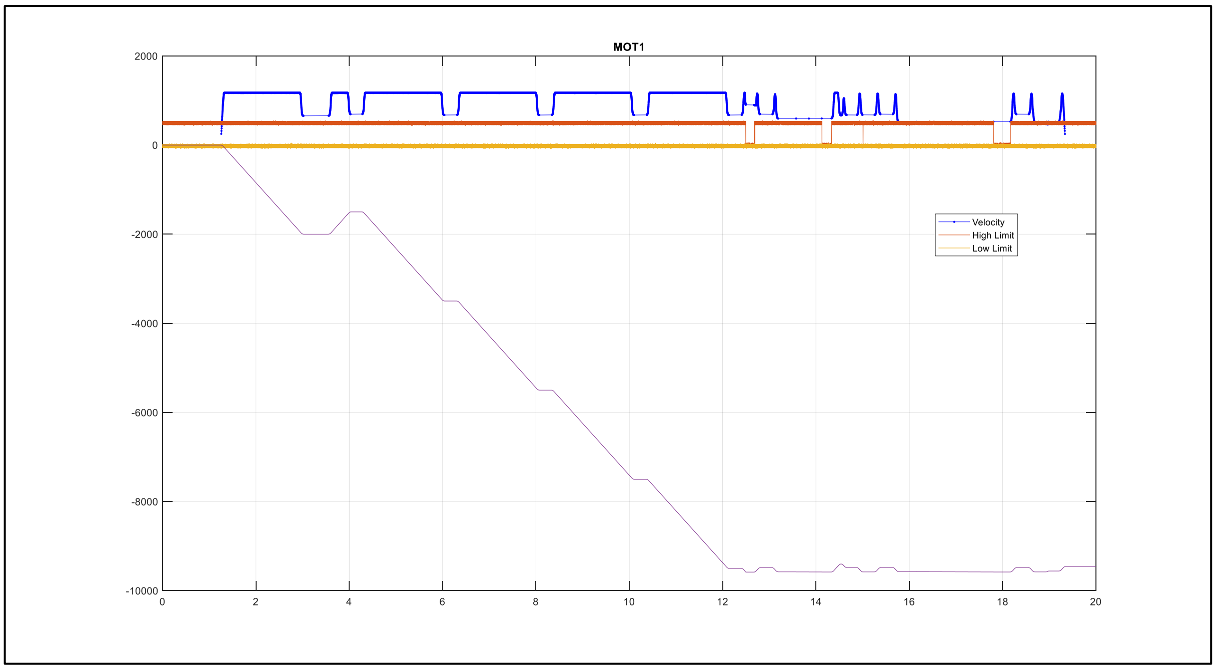


Figure 13 – Position, velocity and switch signals versus time during a movement

All the measurements and experiments that has been explained in Chapter 4 was done for Motor 1 of the impedance tuners during an initialization process. The initialization of a motor is a predefined movements sequence controlled by the old tuner controller, and it sets the motor to a reference point safely. This movement must be done before doing any other movement as there is no reference set if this operation is not done.

The methods, measurements and techniques described throughout Chapter 4 was repeated for all the other motors for the same tuner and another tuner and also repeated with different movement scenarios in order to understand widely and extensively how the old tuner controller controls the motors of the impedance tuners.

4.5 Overall Characteristics and Additional Functionalities

It was observed both visually and numerically that the tuner controller both accelerates and decelerates the motors with a predetermined acceleration value. Moreover, there exists a maximum velocity value which can be seen in Figure 13. All these values were noted down to use again during the development of new controller. For a sufficient movement, the motor controller tries to set the velocity to that maximum value with the defined acceleration. If the movement is not long enough for the motors to accelerate to the maximum velocity, the spikes – which can be seen to the right of the velocity trace in Figure 13- occur. Furthermore, regardless of the movement target, the movement is done with 2000 micro-step intervals. At every 2000 micro-steps, controller stops the motors.

Table 1: Motor performance parameters for impedance tuner MT982A02

	Acceleration	Velocity	Run Current	Idle Current
Motor 1	14861 steps/s^2	1180 steps/s	0.24A	0.06A
Motor 2	22263 steps/s^2	4860 steps/s	0.28A	0.035A
Motor 3	22299 steps/s^2	4860 steps/s	0.28A	0.035A

4.6 Backlash Compensation

An important phenomenon called “Backlash compensation” was also noticed. Since these stepper motors create their movement via gears, looseness and axis sliding of gears can easily happen. This unintentional movements can change the desired RF characteristics significantly. As a matter of fact, this was verified throughout the testing phase of the new controller and will be discussed in the preceding chapters. To overcome this problem, old tuner controller always stops the motors before hitting the target and moves a predefined number of steps more to reach to the desired target. This value was noted as 100 half-steps and can be seen in the right side of the position trace in the Figure 13. To exemplify this, if the target position is 2000 full-steps, the motor controller moves the motor to position 1900, stops and then moves it to 2000. By doing this, every position reached by the motor is guaranteed to have backlash compensation. This logic was also implemented to the new controller in a better way.

After completing the characterization of the old tuner controller, the development and crafting of the new tuner controller was the next task.

5.DEVELOPMENT OF NEW TUNER CONTROLLER

For the development of the new tuner, TRINAMIC Motion Control board TMCM-6213 has been used. This board can control stepper motors up to 6 axes. Furthermore, digital signals can be processed and utilized with this board. This property has been manipulated in the most efficient way in order the enhance the tuner controller with best qualifications.

5.1 Hardware Development of New Tuner Controller

There were specifically two models of RF impedance tuners which were desired to be controlled by the tuner controller. The motors of these impedance tuners were analysed in detail since they were the motors to be controlled. There were two types of motors inside the tuner. Phytron ZS32 and PX234 They can be seen in the figure below.

Figure 14 – PX234

Figure 15 – Phytron ZS32

Furthermore, there were two light barrier sensors which were described in chapter 4.3 for each motor and these sensors were installed with a basic PCB structure inside the RF impedance tuner. In total, the signals for controller board to process were phase current signals and light barrier switch signals. Wires for these signals were connected to designated pins of the TRINAMIC board.

In order to match these outputs of the motor controller board with the pinout of the RF impedance controller, there a PCB was designed. The modifications with respect to signal polarities and phase currents has been done. The one can see the mentioned PCB in the figure below.

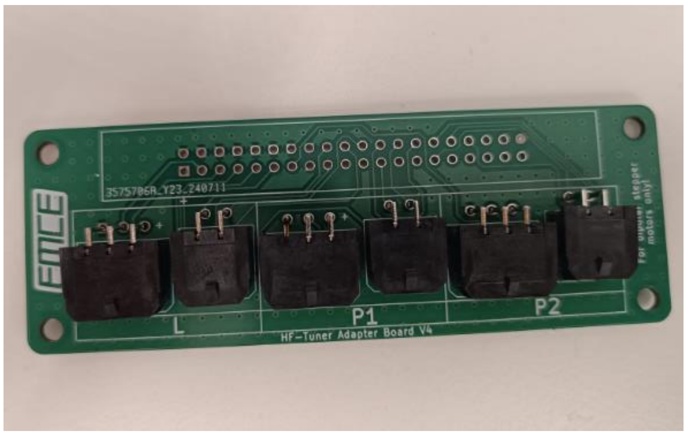


Figure 16 – Connector PCB

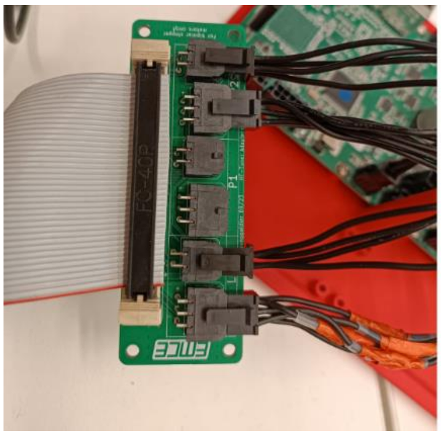


Figure 17 – Cables connected

With the help of this PCB, controller was fully able to send and receive signals to the sensor and motors inside the impedance tuner. The configuration inside the tuner controller can be seen in the figures below.

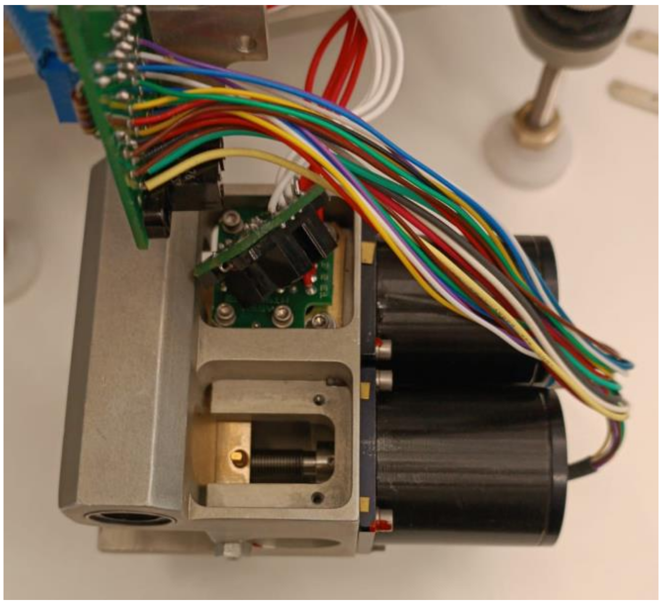


Figure 18 – Two motors (black, on right) and light sensor (black, on the left) inside impedance tuner

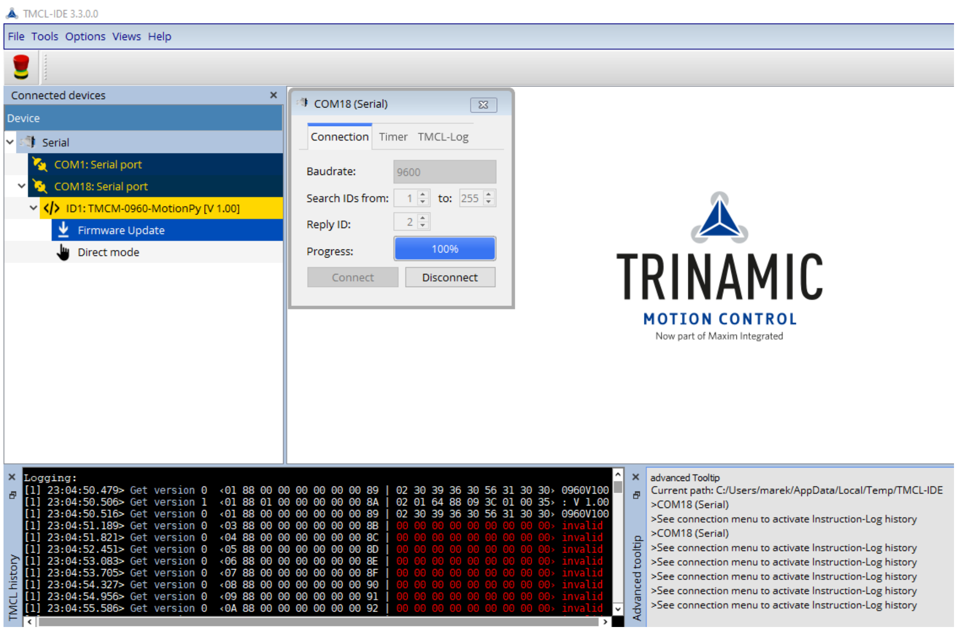
The tuner controller was able to control two tuners at the same time, therefore two ports were installed to the front panel of the controller. Furthermore, an LED which indicates the power on and two LEDs which indicates the intialization and movements of the motors were also applied to the front panel. This can be observed in Figure 31.

The new controller was connected to DC supply throughout the development prcoess. After obtaining the final configuration, a 24V DC power adapter was used and the power was directly drawn from the power socket. This voltage value was sufficient for the controlling process. During the development, it has been observed that voltage values lower than this value can cause some problems during the movement of the motors. The power port can be seen in Figure 32.

5.2 Software Development of New Tuner Controller

As it has been emphasized numerously in the previous sub-chapter, the only components to control or to communicate were light barrier sensors and stepper motors. Furthermore, by knowing the fact that TRINAMIC motor controller board was a sufficient and enhanced board, there were not so much need for complex hardware components. The next task was to implement the controlling process into the board in a desired way.

TRINAMIC motor controller board had a software interface in order to connect to the board and send commands or read data. This interface was communicating with the firmware inside the board, which was called TMCL. Furthermore, there was an IDE for this firmware to code with a specific syntax called TMCL-IDE. With the help of this IDE some basic pre-defined operations could be executed on the board from GUI.



Baudrate: 9600
Search IDs from: 1 to: 255
Reply ID: 2
Progress: 100%

```
Logging:
[1] 23:04:50.479> Get version 0  <01 88 00 00 00 00 00 00 89 | 02 30 39 36 30 56 31 30 30> 0960V100
[1] 23:04:50.506> Get version 1  <01 88 01 00 00 00 00 00 8A | 02 01 64 88 09 3C 01 00 35> : V 1.00
[1] 23:04:50.516> Get version 0  <01 88 00 00 00 00 00 00 89 | 02 30 39 36 30 56 31 30 30> 0960V100
[1] 23:04:51.189> Get version 0  <03 88 00 00 00 00 00 00 8B | 00 00 00 00 00 00 00 00 00> invalid
[1] 23:04:51.821> Get version 0  <04 88 00 00 00 00 00 00 8C | 00 00 00 00 00 00 00 00 00> invalid
[1] 23:04:52.451> Get version 0  <05 88 00 00 00 00 00 00 8D | 00 00 00 00 00 00 00 00 00> invalid
[1] 23:04:53.083> Get version 0  <06 88 00 00 00 00 00 00 8E | 00 00 00 00 00 00 00 00 00> invalid
[1] 23:04:53.705> Get version 0  <07 88 00 00 00 00 00 00 8F | 00 00 00 00 00 00 00 00 00> invalid
[1] 23:04:54.327> Get version 0  <08 88 00 00 00 00 00 00 90 | 00 00 00 00 00 00 00 00 00> invalid
[1] 23:04:54.956> Get version 0  <09 88 00 00 00 00 00 00 91 | 00 00 00 00 00 00 00 00 00> invalid
[1] 23:04:55.586> Get version 0  <0A 88 00 00 00 00 00 00 92 | 00 00 00 00 00 00 00 00 00> invalid
```

```
advanced Tooltip
Current path: C:/Users/marek/AppData/Local/Temp/TMCL-IDE
>COM18 (Serial)
>See connection menu to activate Instruction-Log history
>COM18 (Serial)
>See connection menu to activate Instruction-Log history
>See connection menu to activate Instruction-Log history
>See connection menu to activate Instruction-Log history
>See connection menu to activate Instruction-Log history
>See connection menu to activate Instruction-Log history
```

Figure 19 – TMCL IDE

5.2.1 Introduction to TMCL IDE and TMCL syntax

To be able to explain the methods and approaches which were used in the main software code, the explanation of TMCL firmware is a must. In this chapter, TMCL IDE, board firmware, built-in functions and memory elements of the board will be introduced for future proficiency. Details about the firmware can be found in the reference and datasheets at the end of the document.

The IDE was communicating with the board via an USB cable in a serial communication fashion. Furthermore, there was a specially designed command message packet format – which has been manipulated for advanced purposes and will be mentioned in following chapters.

TMCL Command Format	
Bytes	Meaning
1	Module address
1	Command number
1	Type number
1	Motor or Bank number
4	Value (MSB first!)
1	Checksum

Figure 20 – Command Format

TMCL Reply Format	
Bytes	Meaning
1	Reply address
1	Module address
1	Status (e.g. 100 means no error)
1	Command number
4	Value (MSB first!)
1	Checksum

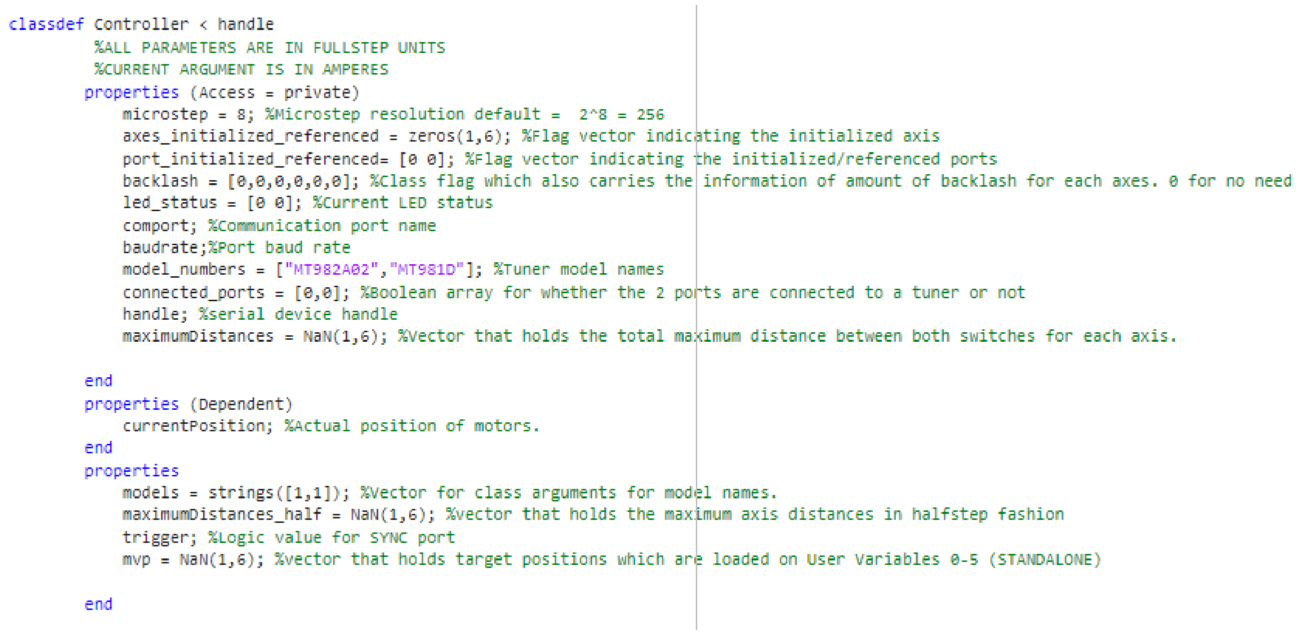
Figure 21 – Reply Format

There is an EEPROM, and a RAM built inside the board for memory purposes. The EEPROM can hold some amount of data. Nevertheless, the reading and writing operations done onto EEPROM shortens the lifetime of the EEPROM. To build a lasting and sustainable device, all these R/W operations on EEPROM was minimized and the algorithm was designed in a fashion with respect to this problem. This also brought some challenges and problems, and different algorithms were utilized in order to overcome these problems.

TMCL-IDE which can be seen in Figure 19 was an UI tool which eases the process of sending commands to and reading data from the motors. There were basic parameters such as maximum speed, maximum acceleration to determine for the motors in the IDE. These were called "axis parameters" and can be configured arbitrarily. Furthermore, phase currents, actual positions and actual velocities could be monitored in real-time with from the IDE. The IDE was a generic tool to control the motors. However, the requirements and expectations from the new tuner was more specific. Therefore, a special approach using the TMCL command packages were utilized. A MATLAB class was developed solely for this purpose. This will be described in detail in the following chapter.

6. MATLAB Controller Class

Although the commercial tool for the board which was explained in chapter 5 was more than sufficient to control the tuners, it was a generic tool, and it was full of built-in functions. To customize and implement specific process and functions, there was a need for a custom controller software. To achieve this, a MATLAB class called "Controller" was developed. The class was designed with respect to requirements of the new board. This can easily be considered as the "customized" firmware inside the new tuner controller. In this chapter, this class will be introduced in sub-chapters.

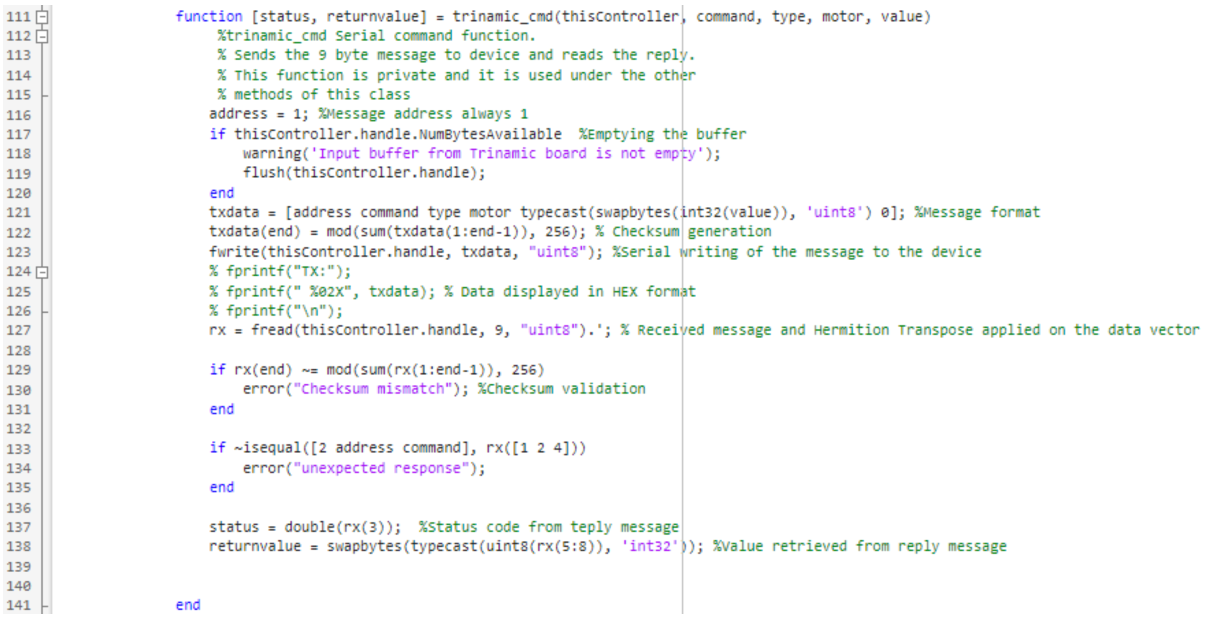


```
classdef Controller < handle
         %ALL PARAMETERS ARE IN FULLSTEP UNITS
         %CURRENT ARGUMENT IS IN AMPERES
        properties (Access = private)
            microstep = 8; %Microstep resolution default =  2^8 = 256
            axes_initialized_referenced = zeros(1,6); %Flag vector indicating the initialized axis
            port_initialized_referenced= [0 0]; %Flag vector indicating the initialized/referenced ports
            backlash = [0,0,0,0,0,0]; %Class flag which also carries the information of amount of backlash for each axes. 0 for no need
            led_status = [0 0]; %Current LED status
            comport; %Communication port name
            baudrate;%Port baud rate
            model_numbers = ["MT982A02","MT981D"]; %Tuner model names
            connected_ports = [0,0]; %Boolean array for whether the 2 ports are connected to a tuner or not
            handle; %serial device handle
            maximumDistances = NaN(1,6); %Vector that holds the total maximum distance between both switches for each axis.

        end
        properties (Dependent)
            currentPosition; %Actual position of motors.
        end
        properties
            models = strings([1,1]); %Vector for class arguments for model names.
            maximumDistances_half = NaN(1,6); %vector that holds the maximum axis distances in halfstep fashion
            trigger; %Logic value for SYNC port
            mvp = NaN(1,6); %vector that holds target positions which are loaded on User Variables 0-5 (STANDALONE)

        end
```

Figure 22 – Controller class in MATLAB

6.1 Communication with the TRINAMIC Board

The message properties described in Figure 20 and Figure 21 were manipulated to design this class and add functionalities. A function named "trinamic_cmd" was created to convert MATLAB level commands to serial commands in the specified message format. This function was the most essential one in the whole class since all the commands were sent by using this function. In other words, all the other high-level functions within the MATLAB class were utilizing this fundamental function.



```matlab
        function [status, returnvalue] = trinamic_cmd(thisController, command, type, motor, value)
            %trinamic_cmd Serial command function.
            % Sends the 9 byte message to device and reads the reply.
            % This function is private and it is used under the other
            % methods of this class
           address = 1; %Message address always 1
           if thisController.handle.NumBytesAvailable  %Emptying the buffer
               warning('Input buffer from Trinamic board is not empty');
               flush(thisController.handle);
           end
           txdata = [address command type motor typecast(swapbytes(int32(value)), 'uint8') 0]; %Message format
           txdata(end) = mod(sum(txdata(1:end-1)), 256); % Checksum generation
           fwrite(thisController.handle, txdata, "uint8"); %Serial writing of the message to the device
           % fprintf("TX:");
           % fprintf(" %02X", txdata); % Data displayed in HEX format
           % fprintf("\n");
           rx = fread(thisController.handle, 9, "uint8").'; % Received message and Hermition Transpose applied on the data vector

           if rx(end) ~= mod(sum(rx(1:end-1)), 256)
               error("Checksum mismatch"); %Checksum validation
           end

           if ~isequal([2 address command], rx([1 2 4]))
               error("unexpected response");
           end

           status = double(rx(3));  %Status code from teply message
           returnvalue = swapbytes(typecast(uint8(rx(5:8)), 'int32')); %Value retrieved from reply message


        end
```

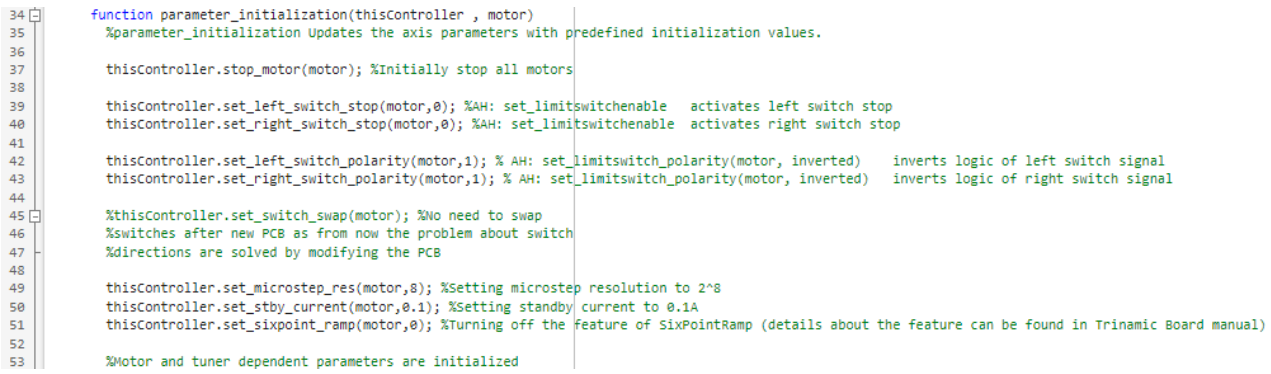
Figure 23 – “trinamic_cmd” function

Function was creating the message packets and sending it to the board in a serial fashion. It was also reading the reply packet, and it also had error handling mechanisms.

6.2 Initialization of the Motors

Initialization is a process where all the motors are moved to a predefined position with predetermined acceleration and velocity values. All this process was done by the old tuner controller and was called “initialization”. Before any other movement, initialization of the motors had to be done. This process was to ensure every movement is starting from the same reference, in order to create reliable and reproducible RF characteristics.

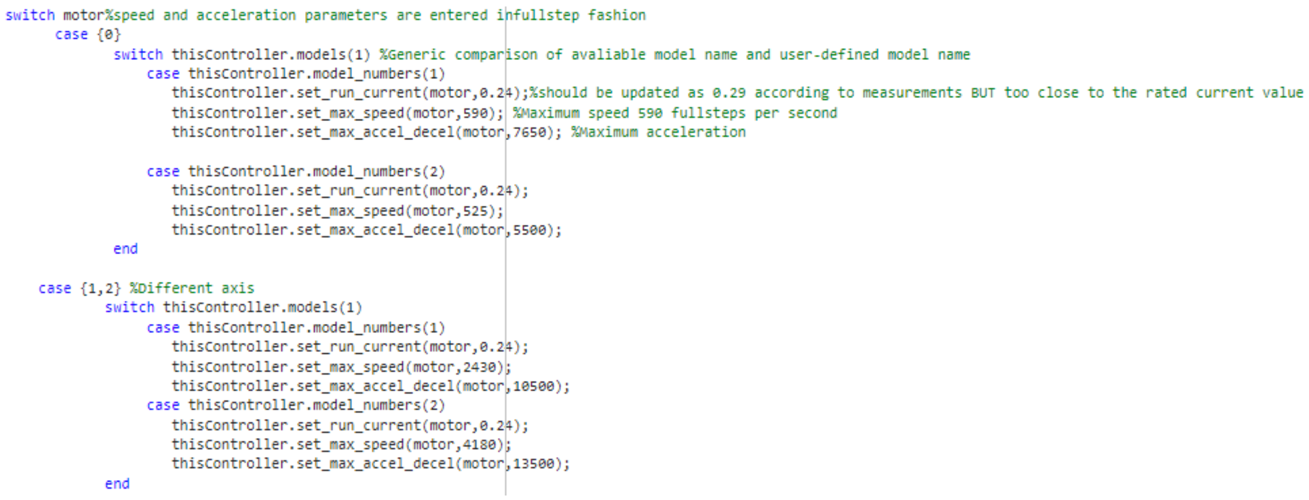
This simple but important process was also repeated in the new tuner controller. Before any other movement, controller class sets the axis parameters such as acceleration, maximum current, maximum velocity to some default values which were obtained during the characterization process and can be found in Table 1.



```matlab
    function parameter_initialization(thisController , motor)
      %parameter_initialization Updates the axis parameters with predefined initialization values.

      thisController.stop_motor(motor); %Initially stop all motors

      thisController.set_left_switch_stop(motor,0); %AH: set_limitswitchenable   activates left switch stop
      thisController.set_right_switch_stop(motor,0); %AH: set_limitswitchenable  activates right switch stop

      thisController.set_left_switch_polarity(motor,1); % AH: set_limitswitch_polarity(motor, inverted)    inverts logic of left switch signal
      thisController.set_right_switch_polarity(motor,1); % AH: set_limitswitch_polarity(motor, inverted)   inverts logic of right switch signal

      %thisController.set_switch_swap(motor); %No need to swap
      %switches after new PCB as from now the problem about switch
      %directions are solved by modifying the PCB

      thisController.set_microstep_res(motor,8); %Setting microstep resolution to 2^8
      thisController.set_stby_current(motor,0.1); %Setting standby current to 0.1A
      thisController.set_sixpoint_ramp(motor,0); %Turning off the feature of SixPointRamp (details about the feature can be found in Trinamic Board manual)

      %Motor and tuner dependent parameters are initialized
```

Figure 24 – parameter_initialization function

This function was setting the microstep resolution, maximum current parameter, maximum speed parameter and maximum acceleration/decelaration parameters to default values. This default values were axis and tuner dependent values. Furthermore, the TRINAMIC board had some built-in functions for a smoother and safer movement of the motors. These functionalities which will be described later were also turned off in the initialization phase to have a full control on the motors.



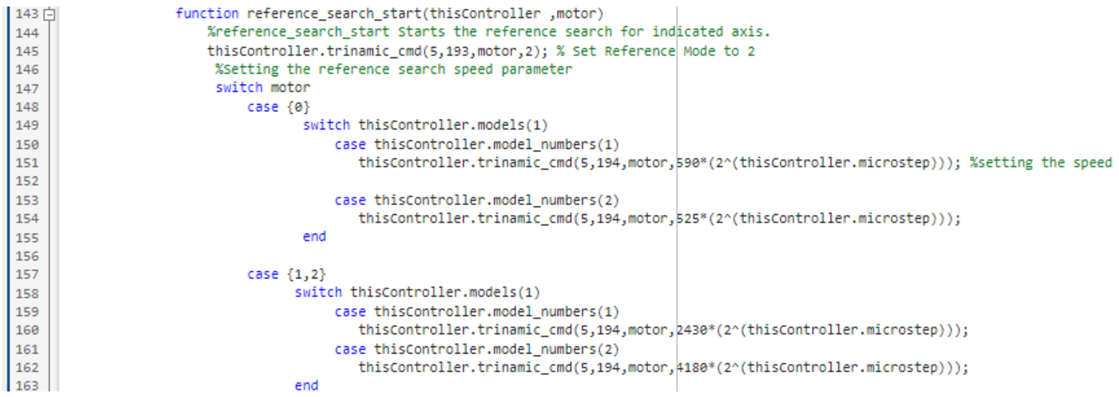
```
switch motor%speed and acceleration parameters are entered infullstep fashion
      case {0}
             switch thisController.models(1) %Generic comparison of avaliable model name and user-defined model name
                 case thisController.model_numbers(1)
                    thisController.set_run_current(motor,0.24);%should be updated as 0.29 according to measurements BUT too close to the rated current value
                    thisController.set_max_speed(motor,590); %Maximum speed 590 fullsteps per second
                    thisController.set_max_accel_decel(motor,7650); %Maximum acceleration

                 case thisController.model_numbers(2)
                    thisController.set_run_current(motor,0.24);
                    thisController.set_max_speed(motor,525);
                    thisController.set_max_accel_decel(motor,5500);
             end

    case {1,2} %Different axis
            switch thisController.models(1)
                 case thisController.model_numbers(1)
                    thisController.set_run_current(motor,0.24);
                    thisController.set_max_speed(motor,2430);
                    thisController.set_max_accel_decel(motor,10500);
                 case thisController.model_numbers(2)
                    thisController.set_run_current(motor,0.24);
                    thisController.set_max_speed(motor,4180);
                    thisController.set_max_accel_decel(motor,13500);
            end
```

Figure 25 – parameter_initialization function continued

There was an in-class check mechanism for this initialization function. In other words, an uninitialized motor was not responding to any movement functions. All the parameter setting operations were done by sending the right message packets and utilizing the function described in chapter 6.1.

The greatest upgrade about the initialization process was that old tuner controller was not able to initialize the motors simultaneously. In the new tuner, thanks to this controller class, an initialization process was done simultaneously for all motors.

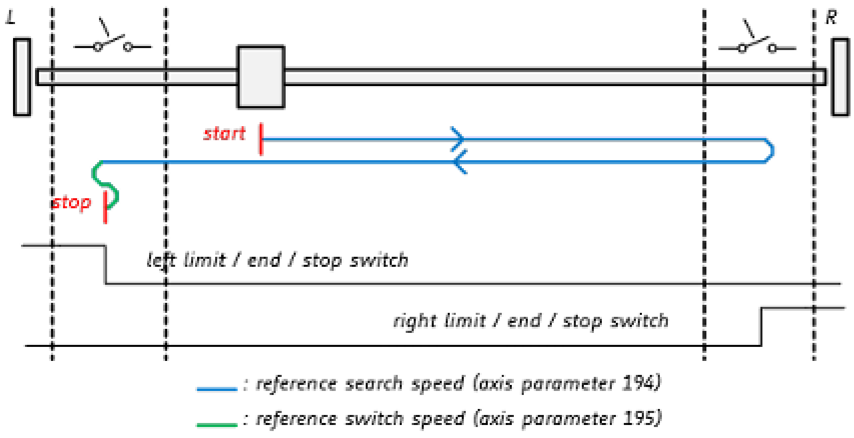
6.3 Reference search



```
function reference_search_start(thisController ,motor)
    %reference_search_start Starts the reference search for indicated axis.
    thisController.trinamic_cmd(5,193,motor,2); % Set Reference Mode to 2
     %Setting the reference search speed parameter
     switch motor
         case {0}
                switch thisController.models(1)
                    case thisController.model_numbers(1)
                       thisController.trinamic_cmd(5,194,motor,590*(2^(thisController.microstep))); %setting the speed

                    case thisController.model_numbers(2)
                       thisController.trinamic_cmd(5,194,motor,525*(2^(thisController.microstep)));
                end

         case {1,2}
               switch thisController.models(1)
                    case thisController.model_numbers(1)
                       thisController.trinamic_cmd(5,194,motor,2430*(2^(thisController.microstep)));
                    case thisController.model_numbers(2)
                       thisController.trinamic_cmd(5,194,motor,4180*(2^(thisController.microstep)));
               end
```

Figure 26 – Function for reference search

To be able to read and write position values to the motor commands, a reference had to be specified. With the help of light barriers, whether a motor hits to an end of the runway can be detected. This detection of two ends is sufficient to obtain the length of a runway in terms of microsteps. In this case a runway is the slab where a motor moves along. This reference search process is also a part of the initialization process as all the motors need a reference point for their movement.

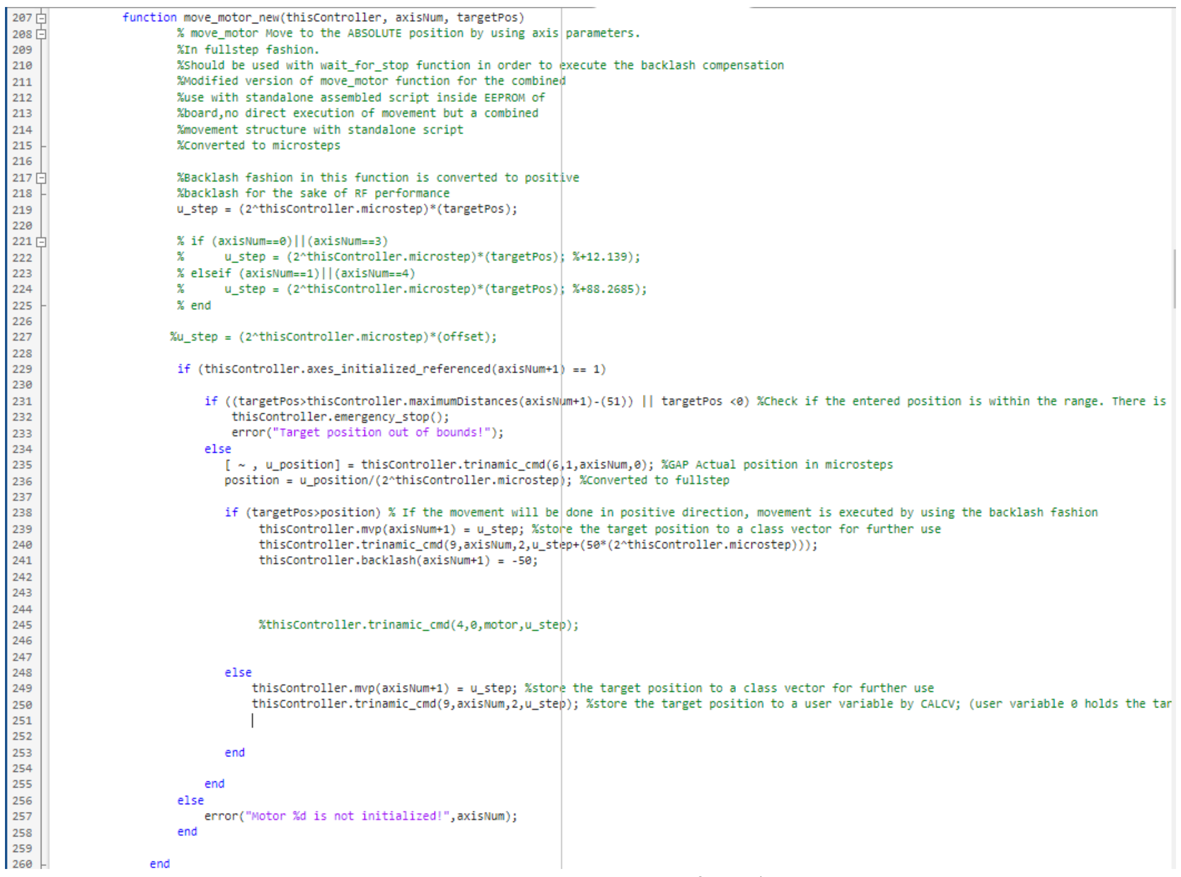


L
R
start
stop
left limit / end / stop switch
right limit / end / stop switch
: reference search speed (axis parameter 194)
: reference switch speed (axis parameter 195)

Figure 27 – Reference search demonstration

6.4 Movement Process

After setting the parameters and determining the range for the motor, a motor is safe to move. As discussed throughout this report, an RF impedance tuner is able to change the impedance thanks to the movement of the motors which changes the reflection. Therefore, movement is the most crucial part of this controlling process. Although there was a ready-to-use movement function, a better movement sequence was developed inside the MATLAB class.



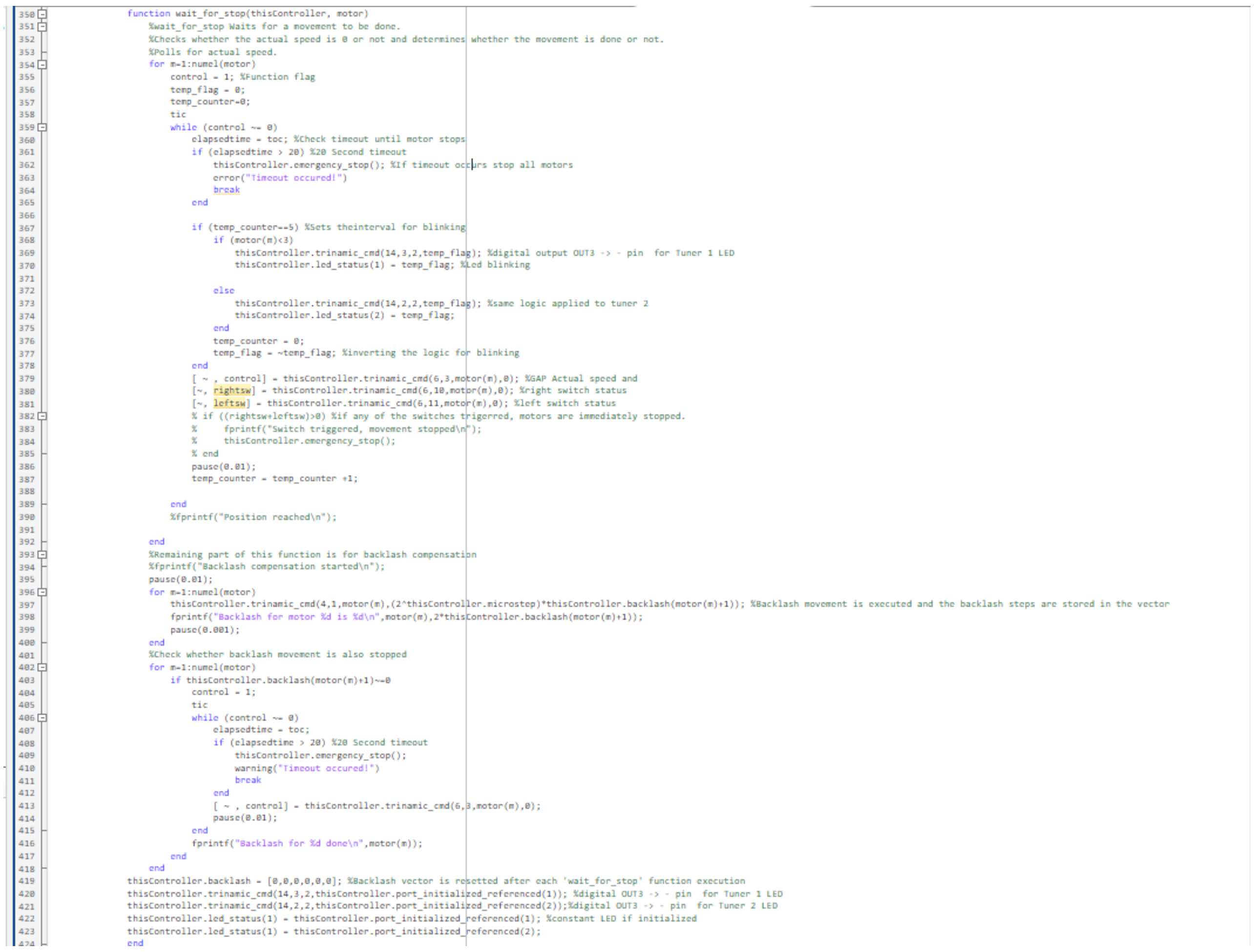
```
        function move_motor_new(thisController, axisNum, targetPos)
            % move_motor Move to the ABSOLUTE position by using axis parameters.
            %In fullstep fashion.
            %Should be used with wait_for_stop function in order to execute the backlash compensation
            %Modified version of move_motor function for the combined
            %use with standalone assembled script inside EEPROM of
            %board,no direct execution of movement but a combined
            %movement structure with standalone script
            %Converted to microsteps

            %Backlash fashion in this function is converted to positive
            %backlash for the sake of RF performance
            u_step = (2^thisController.microstep)*(targetPos);

            % if (axisNum==0)||(axisNum==3)
            %      u_step = (2^thisController.microstep)*(targetPos); %+12.139);
            % elseif (axisNum==1)||(axisNum==4)
            %      u_step = (2^thisController.microstep)*(targetPos); %+88.2685);
            % end

           %u_step = (2^thisController.microstep)*(offset);

            if (thisController.axes_initialized_referenced(axisNum+1) == 1)

                if ((targetPos>thisController.maximumDistances(axisNum+1)-(51)) || targetPos <0) %Check if the entered position is within the range. There is
                    thisController.emergency_stop();
                    error("Target position out of bounds!");
                else
                   [ ~ , u_position] = thisController.trinamic_cmd(6,1,axisNum,0); %GAP Actual position in microsteps
                   position = u_position/(2^thisController.microstep); %Converted to fullstep

                   if (targetPos>position) % If the movement will be done in positive direction, movement is executed by using the backlash fashion
                        thisController.mvp(axisNum+1) = u_step; %store the target position to a class vector for further use
                        thisController.trinamic_cmd(9,axisNum,2,u_step+(50*(2^thisController.microstep)));
                        thisController.backlash(axisNum+1) = -50;



                        %thisController.trinamic_cmd(4,0,motor,u_step);


                   else
                       thisController.mvp(axisNum+1) = u_step; %store the target position to a class vector for further use
                       thisController.trinamic_cmd(9,axisNum,2,u_step); %store the target position to a user variable by CALCV; (user variable 0 holds the tar


                   end

                end
            else
                error("Motor %d is not initialized!",axisNum);
            end

        end
```

Figure 28 – Movement function

The challenges that occurred because of avoiding from using EEPROM which was mentioned before arose during the development of this function. The problems were solved by constantly updating the variables inside the RAM of the motor controller board. This function also executes some inside check mechanisms for safety of the motors, for example it interrupts the movement if a motor reaches to a limit.

6.5 Move and Wait for Stop

To provide a reliable movement, one must be sure about whether a moving motor stopped and reached to its target. This was done by constantly polling the actual position and actual velocity of the motor. Furthermore, this logic was combined with the movement function described in chapter 6.4 and a frequently used combined function named “move_and_wait” was created. This was the main movement function that has been used during the control of the stepper motors.



```
function wait_for_stop(thisController, motor)
    %wait_for_stop Waits for a movement to be done.
    %Checks whether the actual speed is 0 or not and determines whether the movement is done or not.
    %Polls for actual speed.
    for m=1:numel(motor)
        control = 1; %Function flag
        temp_flag = 0;
        temp_counter=0;
        tic
        while (control ~= 0)
            elapsedtime = toc; %Check timeout until motor stops
            if (elapsedtime > 20) %20 Second timeout
                thisController.emergency_stop(); %If timeout occurs stop all motors
                error("Timeout occured!")
                break
            end

            if (temp_counter==5) %Sets theinterval for blinking
                if (motor(m)<3)
                    thisController.trinamic_cmd(14,3,2,temp_flag); %digital output OUT3 -> - pin  for Tuner 1 LED
                    thisController.led_status(1) = temp_flag; %Led blinking

                else
                    thisController.trinamic_cmd(14,2,2,temp_flag); %same logic applied to tuner 2
                    thisController.led_status(2) = temp_flag;
                end
                temp_counter = 0;
                temp_flag = ~temp_flag; %inverting the logic for blinking
            end
            [ ~ , control] = thisController.trinamic_cmd(6,3,motor(m),0); %GAP Actual speed and
            [~, rightsw] = thisController.trinamic_cmd(6,10,motor(m),0); %right switch status
            [~, leftsw] = thisController.trinamic_cmd(6,11,motor(m),0); %left switch status
            % if ((rightsw+leftsw)>0) %if any of the switches trigerred, motors are immediately stopped.
            %     fprintf("Switch triggered, movement stopped\n");
            %     thisController.emergency_stop();
            % end
            pause(0.01);
            temp_counter = temp_counter +1;

        end
        %fprintf("Position reached\n");

    end
    %Remaining part of this function is for backlash compensation
    %fprintf("Backlash compensation started\n");
    pause(0.01);
    for m=1:numel(motor)
        thisController.trinamic_cmd(4,1,motor(m),(2^thisController.microstep)*thisController.backlash(motor(m)+1)); %Backlash movement is executed and the backlash steps are stored in the vector
        fprintf("Backlash for motor %d is %d\n",motor(m),2*thisController.backlash(motor(m)+1));
        pause(0.001);
    end
    %Check whether backlash movement is also stopped
    for m=1:numel(motor)
        if thisController.backlash(motor(m)+1)~=0
            control = 1;
            tic
            while (control ~= 0)
                elapsedtime = toc;
                if (elapsedtime > 20) %20 Second timeout
                    thisController.emergency_stop();
                    warning("Timeout occured!")
                    break
                end
                [ ~ , control] = thisController.trinamic_cmd(6,3,motor(m),0);
                pause(0.01);
            end
            fprintf("Backlash for %d done\n",motor(m));
        end
    end
thisController.backlash = [0,0,0,0,0,0]; %Backlash vector is resetted after each 'wait_for_stop' function execution
thisController.trinamic_cmd(14,3,2,thisController.port_initialized_referenced(1)); %digital OUT3 -> - pin  for Tuner 1 LED
thisController.trinamic_cmd(14,2,2,thisController.port_initialized_referenced(2));%digital OUT3 -> - pin  for Tuner 2 LED
thisController.led_status(1) = thisController.port_initialized_referenced(1); %constant LED if initialized
thisController.led_status(1) = thisController.port_initialized_referenced(2);
end
```

Figure 29 – Wait for stop function

This function also controls some peripheral operation such as LED blinking and also communicates with the light barrier sensor. Furthermore, it also tries to implement the backlash logic that has been discussed in detail. In fact, the backlash algorithm takes places during this function. When a motor is close to reach it’s target, the backlash algorithm starts and moves the motor to the final target point. The reason why backlash compensation is an important problem will be discussed and demonstrated in the following chapters.

6.6 Miscellaneous Functions and Overall Controller Class

As explained in detail, setting the motor performance parameters, executing a movement and stopping a motor with backlash compensation were the most important parts of the motor control. The control class has some further functionalities other than the mentioned ones. For example, an emergency stop function was implemented for stopping the motors in emergency. In addition, class itself creates the serial port which makes the connection more stable as the port opens when the class is created and closes when the class is deleted. There are some other details which makes this controller class more stable, reliable and efficient.

This controller class designed from MATLAB was the main script where the algorithm behind the new tuner controller was implemented. After combining this class with the hardware described above, the tuner controller was ready to use. However, the problem was finding the best configuration with respect to the desired RF characteristics. To achieve this, dozens of RF measurements were done on the new tuner controller. In the preceding chapter this will be discussed.

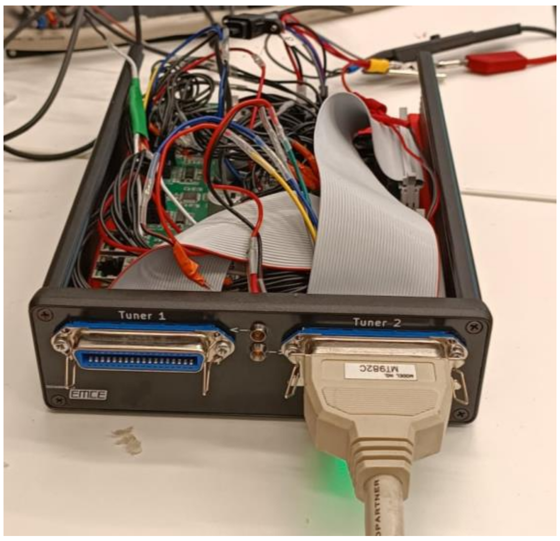


Figure 30 – Interior of tuner controller

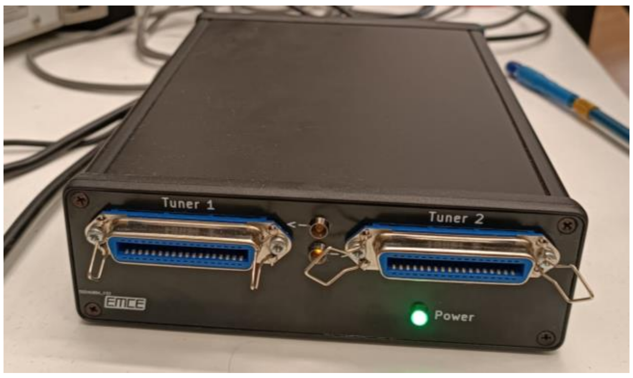


Figure 31 – Front view

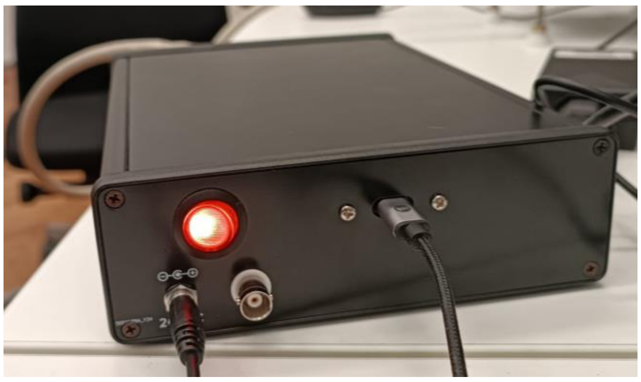


Figure 32 – Rear view

7.RF CHARACTERISTICS OF TUNER CONTROLLER

As stated numerously throughout the report, the main purpose of crafting this new tuner controller was to control the motors inside the tuners. However, a tuner controller should provide logical, correct, reliable and reproducible RF characteristics. In order to maintain this behaviour for the new tuner controller, several RF characterization tests were performed. Some tuning and optimizing operations were done with respect to outcome of these measurements. In this chapter, the test setup will be briefly introduced. Then, the experiments and measurements will be explained.

7.1 Test Setup

The best way to observe the RF characteristics was to obtain the scattering parameters for the tuner controller. In order to measure the S-parameters, a programmable vector network analyzer Agilent PNA E8364A was used. To be able to extend the measurements ports, Agilent S-parameter Test Set ZS623AH50 was used.

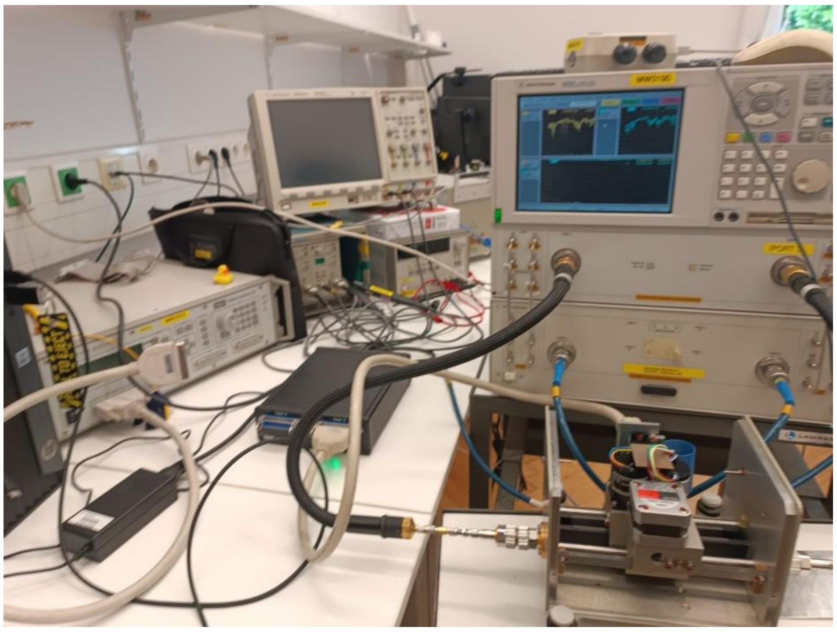


Figure 33 - Network Analyzer

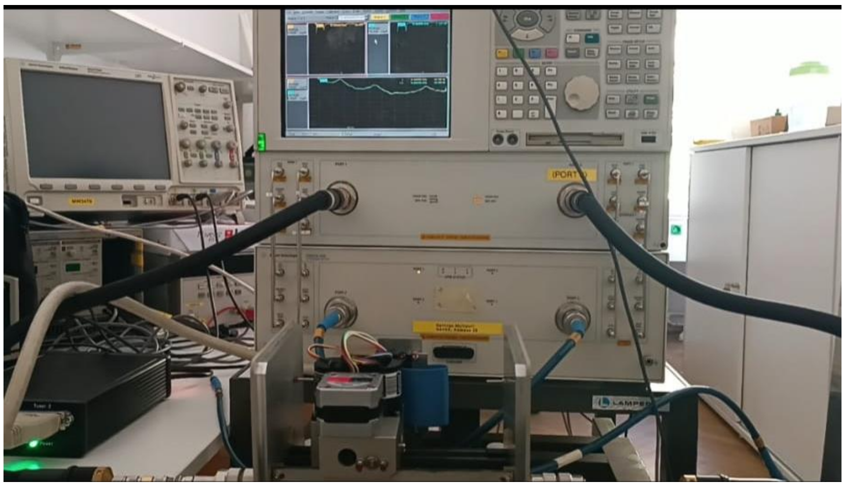
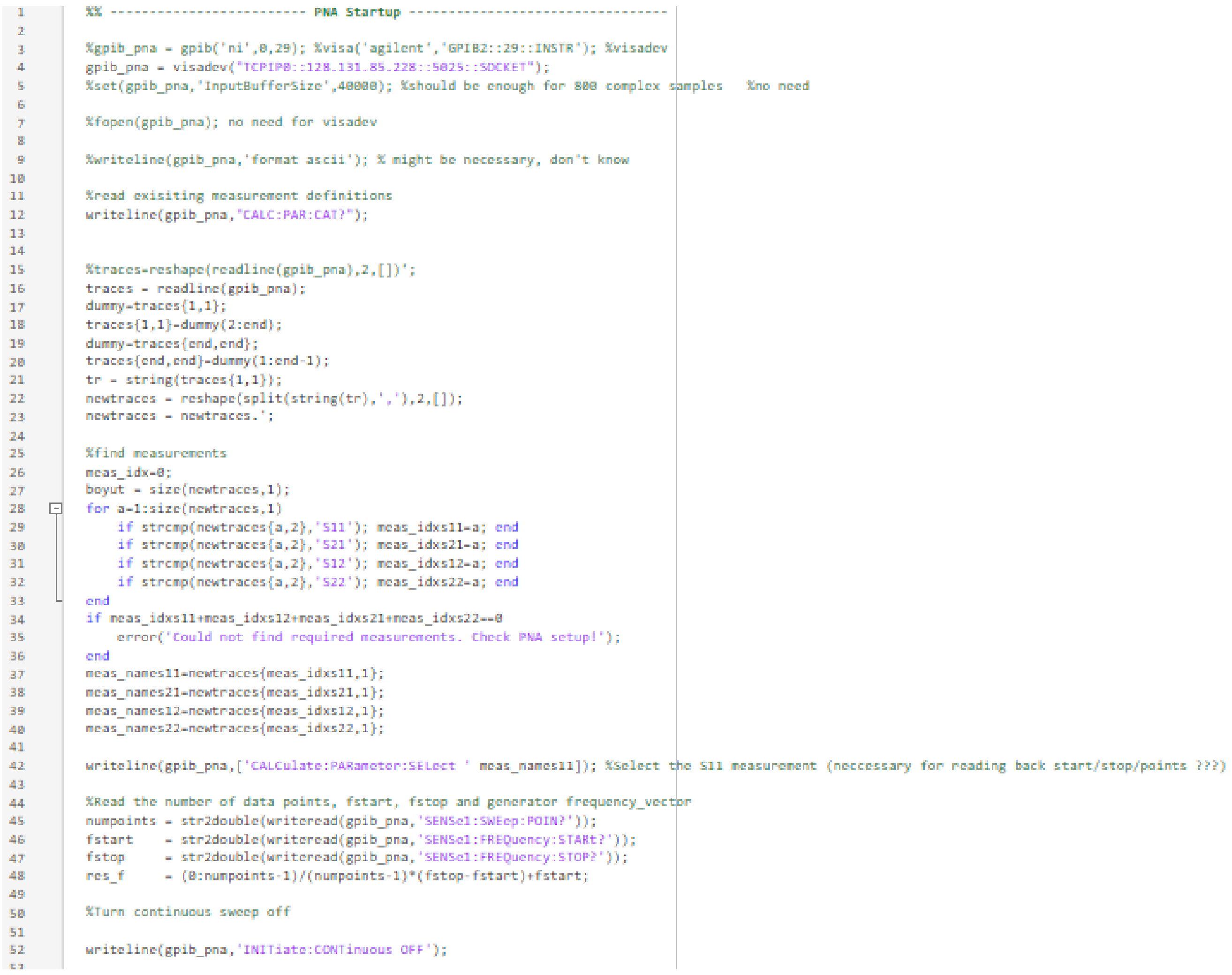


Figure 34 – Test setup, controller connected to VNA

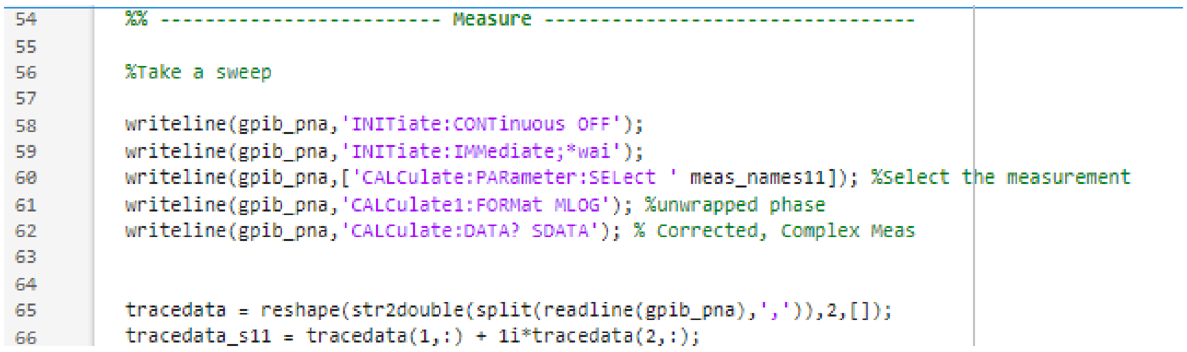
To be sure about the RF characteristics, numerous different movement sequences of the motors were measured. Since the number of measurements and experiments were quite high, an automated approach was a need. By controlling the measurement instruments from a MATLAB script using the VISA/GPIB interfaces, automated measurement sequences were established and utilized.



```
%% ------------------------ PNA Startup --------------------------------

%gpib_pna = gpib('ni',0,29); %visa('agilent','GPIB2::29::INSTR'); %visadev
gpib_pna = visadev("TCPIP0::128.131.85.228::5025::SOCKET");
%set(gpib_pna,'InputBufferSize',40000); %should be enough for 800 complex samples   %no need

%fopen(gpib_pna); no need for visadev

%writeline(gpib_pna,'format ascii'); % might be necessary, don't know

%read exisiting measurement definitions
writeline(gpib_pna,"CALC:PAR:CAT?");


%traces=reshape(readline(gpib_pna),2,[])';
traces = readline(gpib_pna);
dummy=traces{1,1};
traces{1,1}=dummy(2:end);
dummy=traces{end,end};
traces{end,end}=dummy(1:end-1);
tr = string(traces{1,1});
newtraces = reshape(split(string(tr),','),2,[]);
newtraces = newtraces.';

%find measurements
meas_idx=0;
boyut = size(newtraces,1);
for a=1:size(newtraces,1)
    if strcmp(newtraces{a,2},'S11'); meas_idxs11=a; end
    if strcmp(newtraces{a,2},'S21'); meas_idxs21=a; end
    if strcmp(newtraces{a,2},'S12'); meas_idxs12=a; end
    if strcmp(newtraces{a,2},'S22'); meas_idxs22=a; end
end
if meas_idxs11+meas_idxs12+meas_idxs21+meas_idxs22==0
    error('Could not find required measurements. Check PNA setup!');
end
meas_names11=newtraces{meas_idxs11,1};
meas_names21=newtraces{meas_idxs21,1};
meas_names12=newtraces{meas_idxs12,1};
meas_names22=newtraces{meas_idxs22,1};

writeline(gpib_pna,['CALCulate:PARameter:SELect ' meas_names11]); %Select the S11 measurement (neccessary for reading back start/stop/points ???)

%Read the number of data points, fstart, fstop and generator frequency_vector
numpoints = str2double(writeread(gpib_pna,'SENSe1:SWEep:POIN?'));
fstart    = str2double(writeread(gpib_pna,'SENSe1:FREQuency:STARt?'));
fstop     = str2double(writeread(gpib_pna,'SENSe1:FREQuency:STOP?'));
res_f     = (0:numpoints-1)/(numpoints-1)*(fstop-fstart)+fstart;

%Turn continuous sweep off

writeline(gpib_pna,'INITiate:CONTinuous OFF');
```

Figure 35 – Script to automate VNA measurement

With this approach, datasets and traces obtained from measurements were collected and processed for the desired outcomes automatically. The observation and processing about datasets will be explained in the following sub-chapter.



```
%% ------------------------- Measure ---------------------------------

%Take a sweep

writeline(gpib_pna,'INITiate:CONTinuous OFF');
writeline(gpib_pna,'INITiate:IMMediate;*wai');
writeline(gpib_pna,['CALCulate:PARameter:SELect ' meas_names11]); %Select the measurement
writeline(gpib_pna,'CALCulate1:FORMat MLOG'); %unwrapped phase
writeline(gpib_pna,'CALCulate:DATA? SDATA'); % Corrected, Complex Meas


tracedata = reshape(str2double(split(readline(gpib_pna),',')),2,[]);
tracedata_s11 = tracedata(1,:) + 1i*tracedata(2,:);
```

Figure 36 – Measurement of data

7.2 Measurements

The S-parameters were obtained for the tuner controller. However, since there were three different motors inside the tuner, there were so many different cases to measure and analyze.

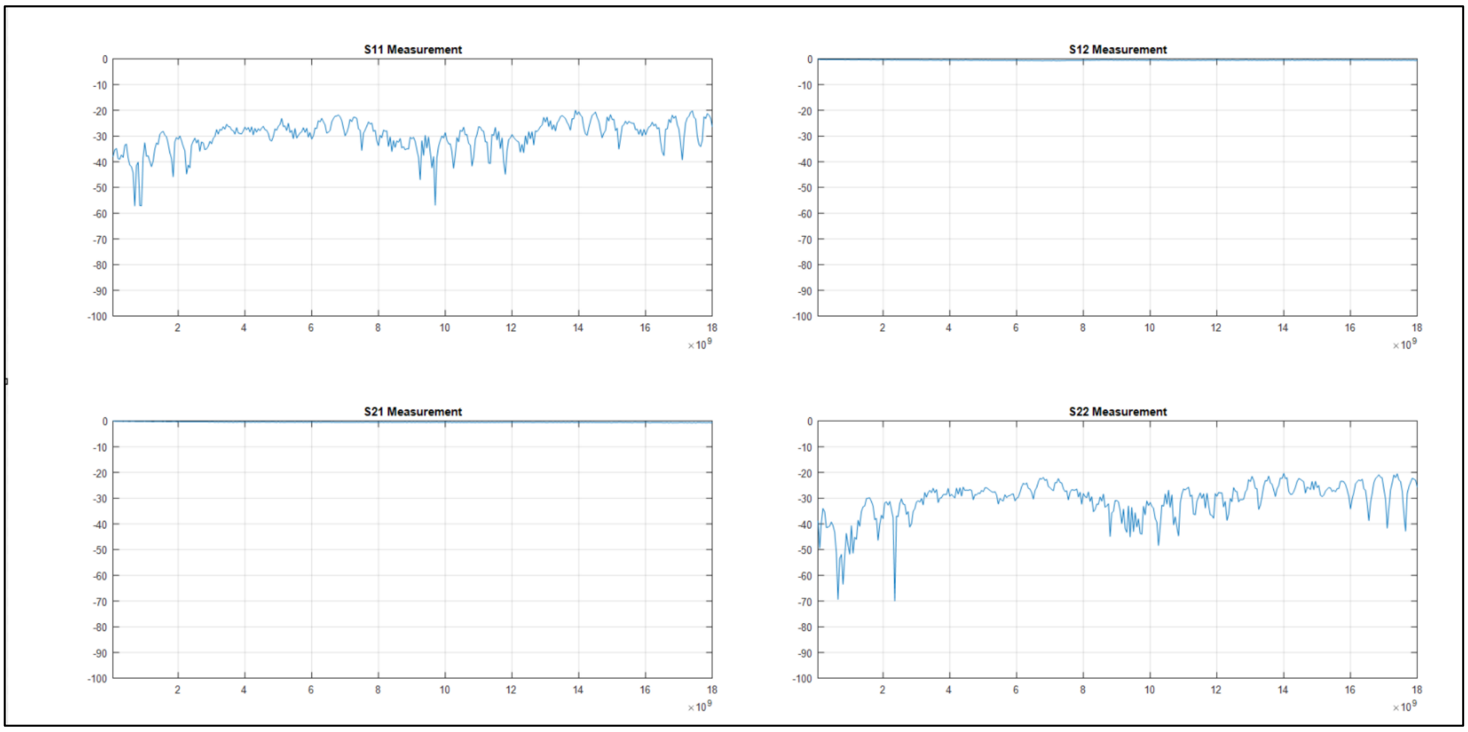


Figure 37 – Scattering parameters

The Maury automated tuners which were planned to be controlled by the new tuner controller had a frequency operation range between 1.8 and 18 GHz. The frequency sweep limits were assigned with respect to this limitation.

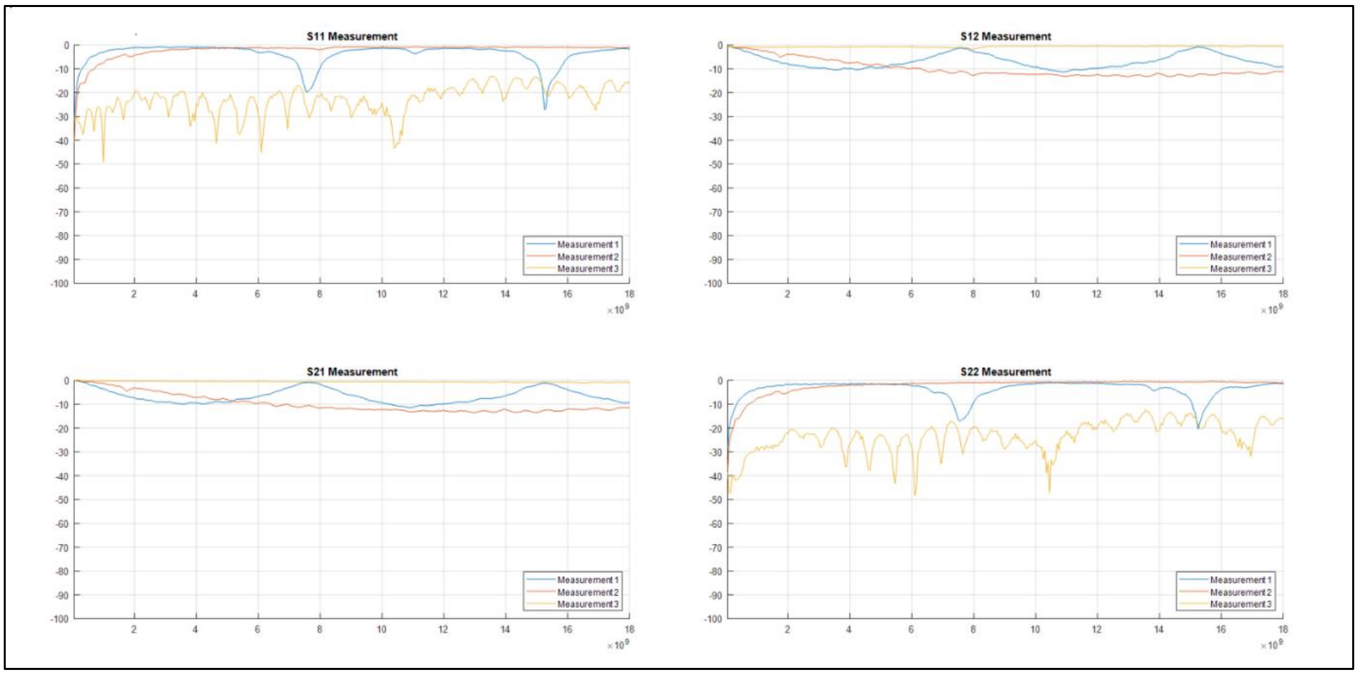


Figure 38 – Scattering parameters for different motor positions

In Figure 38, the one can see different sets of S-parameters. There are three different motor configurations and corresponding different scattering parameters. Movement of some motors only change the magnitude whereas some other change only the phase. As it has been explained in the report, the position of motors determines where the signal reflects. Therefore, measurements involving different motor positions yields different outcomes as expected.

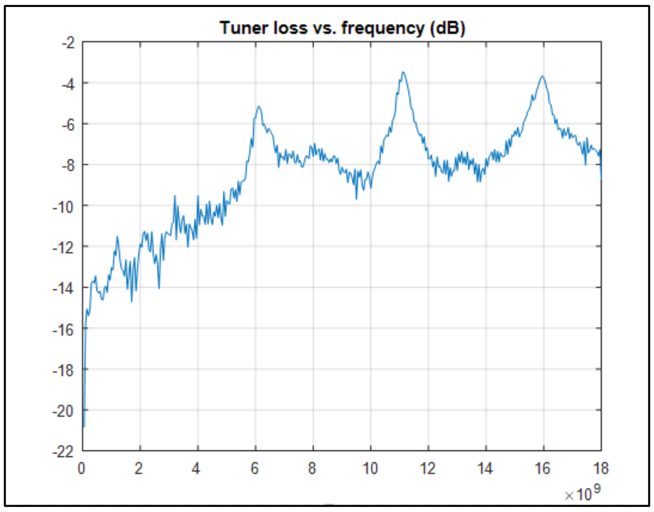


Figure 39 – Tuner loss

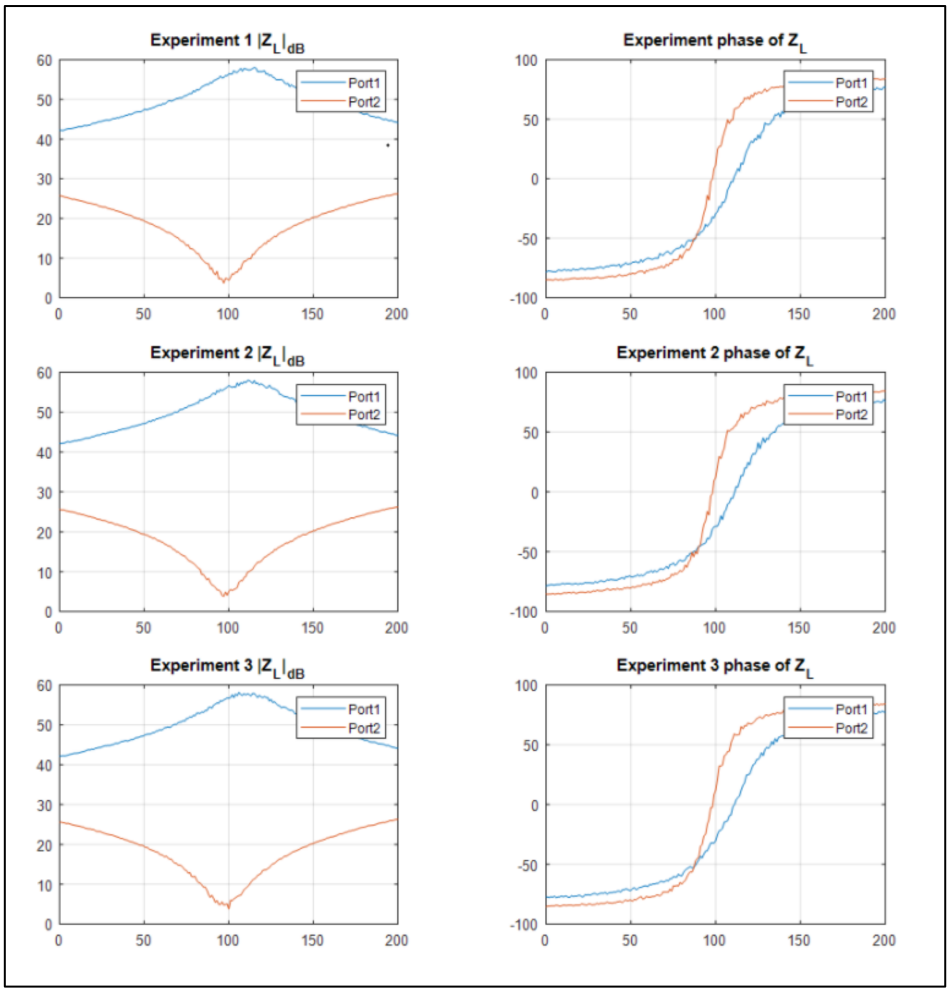


Figure 40 – Impedance characteristics for different motor positions

By using the measured S-parameters, some further information about the tuner controller also obtained such as tuner loss, impedance, etc. The one can observe loss characteristics in Figure 39.

In order to gain insight about RF performance of the new tuner, these measurements were essential but not sufficient. For the new tuner controller to replace the old one, it was expected that new one should provide the necessary RF characteristics. To investigate any possible discrepancies and to optimize and fix them, a comparison between the tuner controllers from the aspect of RF performance was a must. In the following chapter this will be discussed.

8. COMPARISON BETWEEN TUNER CONTROLLERS

8.1 Comparing RF Performances

Through the ends of the internship, the RF performances of two tuner controllers; Maury ATS Controller and the new controller were compared. The task was to replace the existing old tuner controller. Therefore, expectations were that new tuner should be able to provide necessary RF characteristics when compared to the old one.

During the comparison, one small problem about backlash compensation was observed. This will be explained in detail through the ends of this chapter.

To compare RF performances, S-parameters of tuner controllers were compared. However, to enhance the comparison and to be able to compare movements, a frequency was selected. The frequency to compare the RF performances were selected as 18GHz since it was the highest frequency that the impedance tuners can operate on safely. The aim for selecting higher frequency was to observe the RF performances at the most sensitive frequency configuration.

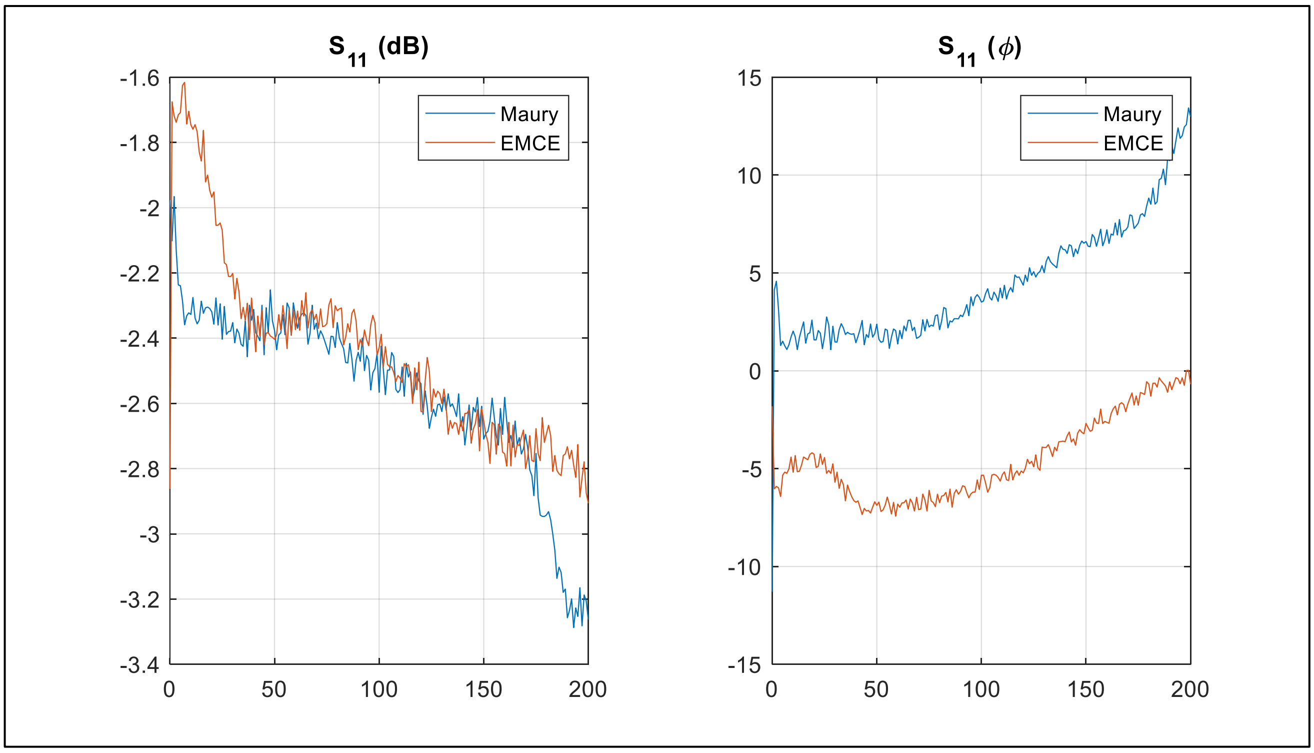


Figure 41 – Comparison of S_{11} parameters

To compare the two controllers from the aspect of RF behaviour, the phase and magnitude of S_{11} parameters at the frequency 18GHz were observed. Further dedications like impedance or tuner loss were easy to obtain from S_{11} and were not so crucial. To observe how the S_{11} changes during a movement of a motor, the motor number 1 has been moved from an absolute position of 0 to 200 by two different controllers. The aim was to repeat same movement process from the two tuner controllers and to observe how similar the RF performance was. To keep the experiment independent from other parameters, the absolute position of motor number 2 and motor number 3 were kept the same for each of the measurements. In the Figure 41, the one can see how the S_{11} changes from position 0 to position 200. The experiment in the figure only consists of the movement of the motor number 1. Further experiments by changing the moving motors, changing the absolute position of halting motors etc. were done to collect more data.

The observation was that, although both controller exhibits slightly similar patterns, it was like as if there was an offset between the traces for parameter S_{11}. This situation has been observed for the other measurement and experiment cases.

Old controller was like a black-box device. The main algorithm behind the motor control was not explicitly explained. To overcome this and to comprehend the logic, all the characterization measurements were done as deeply discussed in Chapter 4. Because of this, the main reason behind the discrepancy demonstrated in Figure 41 was hard to find. It could be many things like motor parameters, duty cycle difference etc. Rather than finding the problem, tuning and optimizing the new tuner controller to match with the RF performance of the old controller was done. Remaining parts of this report explains how this discrepancy is tried to be eliminated.

8.2 Offset Matching

First approach was to find the correct offset in terms of motor position to match the RF characteristics. This approach was tried because of the observations done in chapter 8.1. To find the correct offset, a set of motor configurations with possible offsets values were measured.

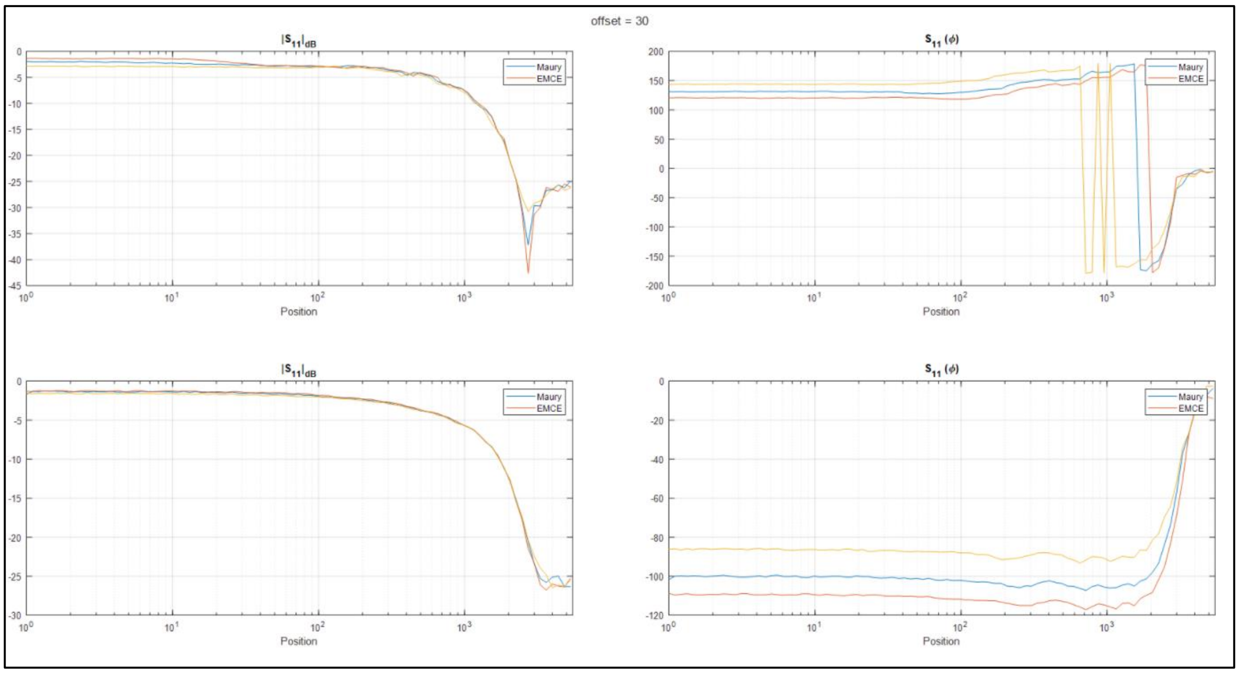


Figure 42 – S_{11} measured with offset of 30 steps

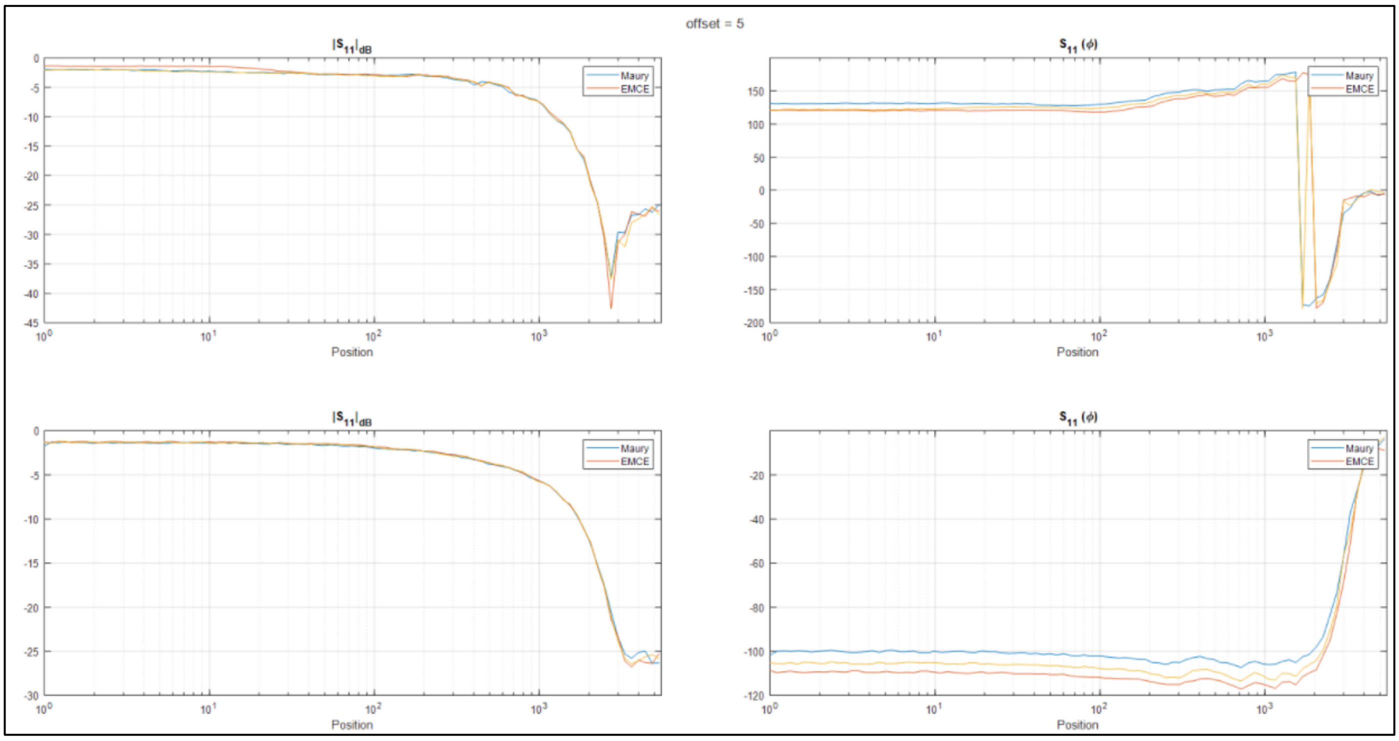


Figure 43 – S_{11} measured with offset of 5 steps

In Figure 42 and Figure 43, the one can see the S_{11} characteristics for different offset values. Furthermore, the two magnitude and phase plot below has been measured when motor 2 and motor 3 moved to a different position than in the figures above. This has been done to see whether these possible offset values work for every motor configuration. However, this approach was slow and as it can be seen from the figures, not very helping. This process also has been automated to try more candidate offsets.

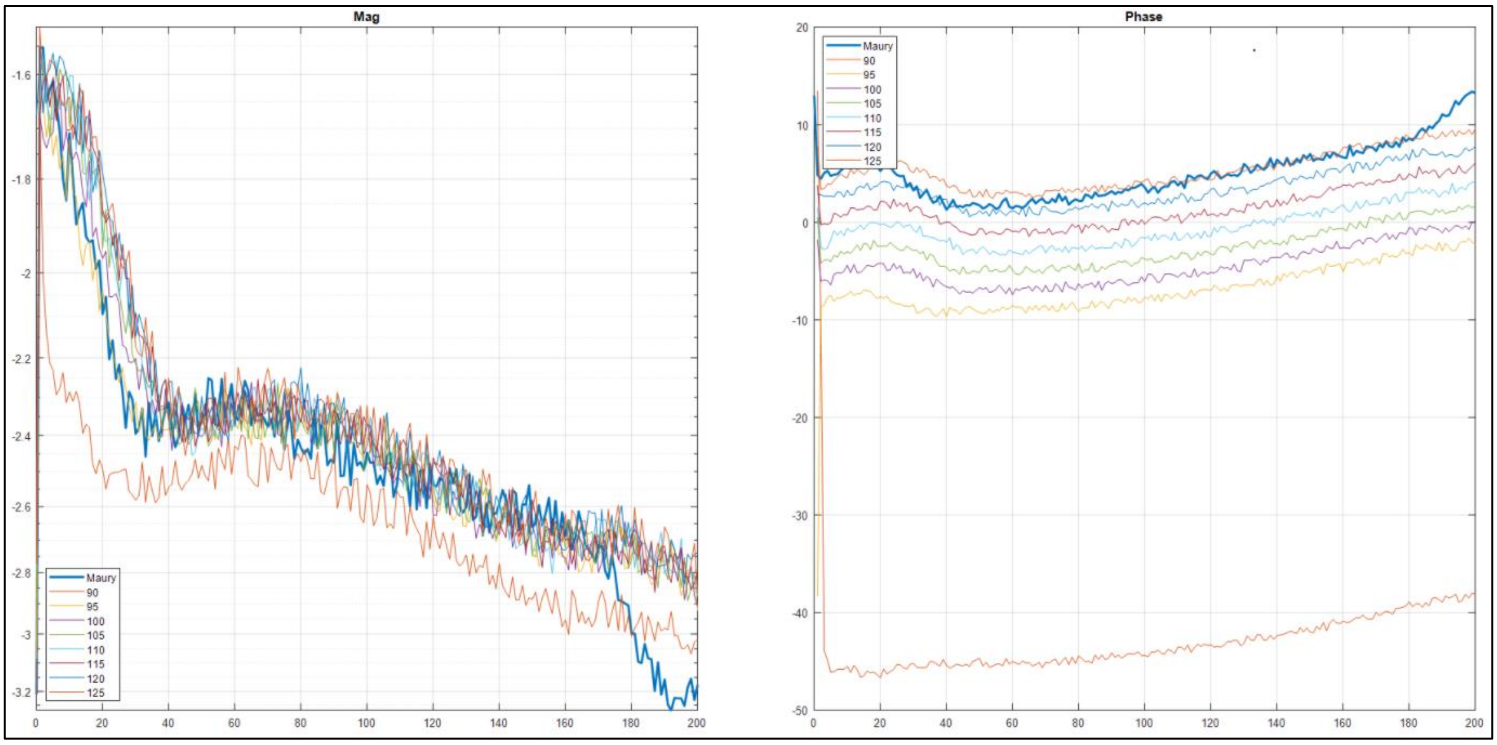


Figure 44 - S_{11} with different offset values

In Figure 44, magnitude and phase of S_{11} during the movement of motor 1 from absolute position of 0 to 200 with different offset values can be seen. With the help of this approach, it could be seen that an offset of 22 motor full-steps was solving the problem.

Although this report generally explains and demonstrates the movement of motor number 1 while motor 2 and motor 3 are constant, all the observations and deductions are repeated for other cases like movement of motor 2 while motor 1 and motor 3 stays constant. For the ease of understanding and for a straight to the point explanation, only the first case is discussed.

8.3 Further Investigations

To test and be sure about the RF performance of the new tuner under all conditions, several more measurements were done. Different motor position configurations, different movement sequences were tested for reliability and reproducibility. Furthermore, some experiments were done to understand the effect of the concept of backlash compensation in a better way. The RF outcome of these different experiments, which were implicitly mentioned throughout the report, will be demonstrated in this sub-chapter.

To ensure that the RF characteristics were reproducible, several experiments were done. For example, motor was moved to a target position from different initial positions. Furthermore, the effect of backlash compensation was also tried to be observed.

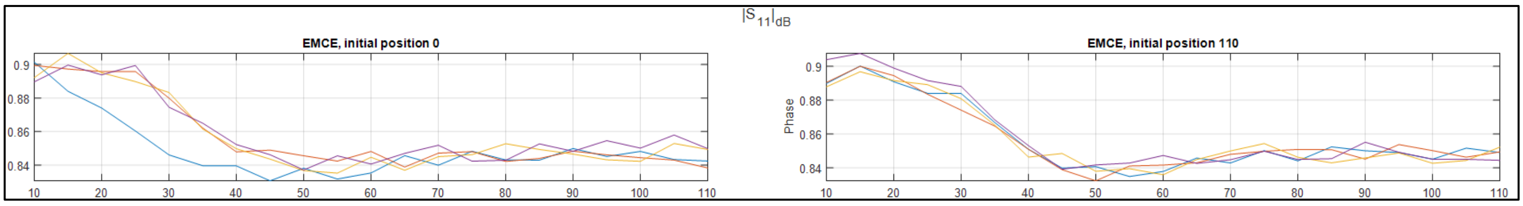


Figure 45 – Magnitude of S_{11} for experiment 1

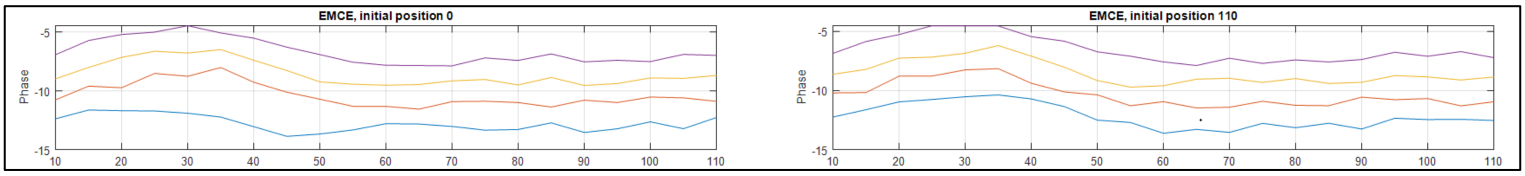


Figure 46 – Phase of S_{11} for experiment 1

Experiment 1 was to move motor 2 to absolute positions of 10,11,12…110 from different initial positions namely 0 and 110 and keeping the other motors constant. From Figure 45 and 46, especially for the lower position values, there were slight differences when approaching from position 0 and approaching from position 110. However, in order to provide a reliable and reproducible RF performance, the tuner controller should demonstrate the same RF performance at the target position independent from the initial position.

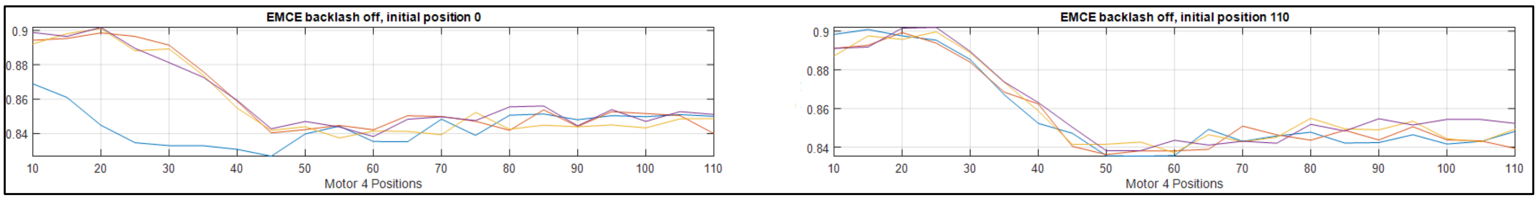


Figure 47 – Magnitude of S_{11} for experiment 2

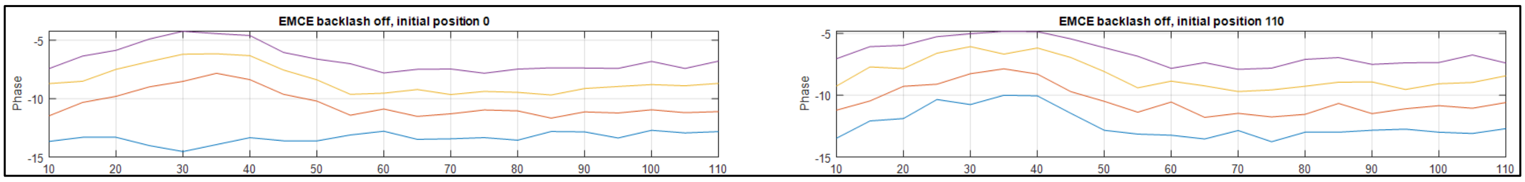


Figure 48 – Phase of S_{11} for experiment 2

It has been thought that maybe the backlash compensation implemented to the new motor controller was incorrect. To check that, experiment 1 was repeated without the backlash compensation feature. The results were same with the experiment 1 and discrepancy was continuing. It was thought that the new tuner controller was not able to provide reliable RF performance unlike the old controller. Another experiment was done by using the Maury controller and repeating experiment 1 to see the correct performance.

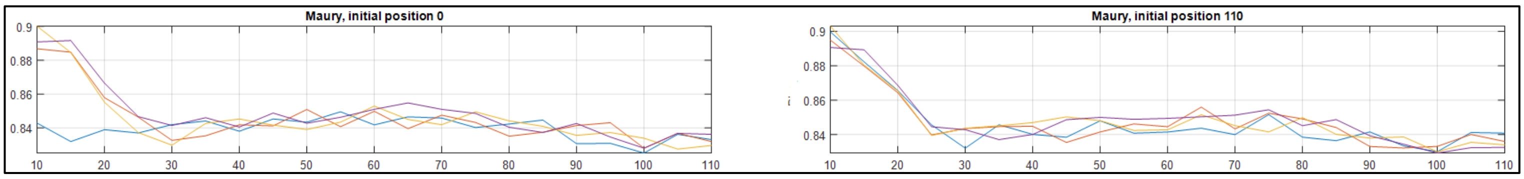


Figure 49 – Magnitude of S_{11} for experiment 3

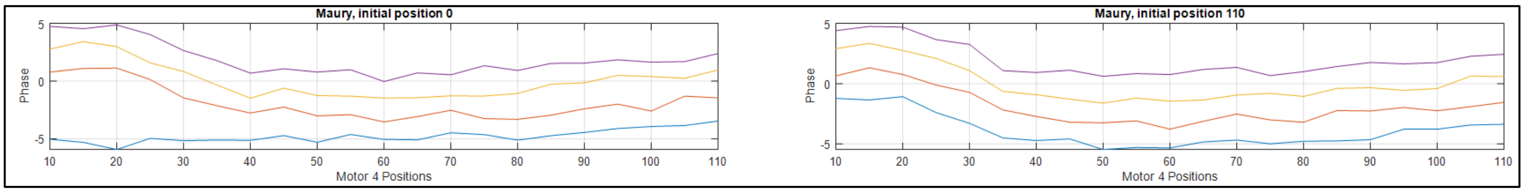


Figure 50 – Phase of S_{11} for experiment 3

It was observed that Maury tuner controller also exhibited the same problem just like the new controller. This was an unexpected result because Maury controller had the backlash compensation algorithm just to avoid this problem. This can also be verified from Figure 49 and 50.

Up until then, problem was thought to be the wrong backlash implementation in the new tuner controller. After observing the old tuner controller and seeing the same result, it was seen that the new controller had the correct implementation and it was yielding the same performance with the old one. However, still the approaching direction was changing the S-parameters, and this was a big problem in terms of reproducibility.

8.4 Backlash Compensation Problem and Solution

When designing the controller class, backlash compensation algorithm was implemented (Chapter 6.5). However, this backlash compensation was only designed to work for a single direction. In other words, if motor was moving to a lower position, backlash compensation was not done. The reason for this was the observations made during the characterization of the old tuner controller (Chapter 4). However, with the latest measurement it has been seen that the existing backlash algorithm was not sufficient.

To overcome this discrepancy, backlash compensation was updated and designed as it works for movements for both directions. For every target position, motor was reaching to the final position with backlash compensation.

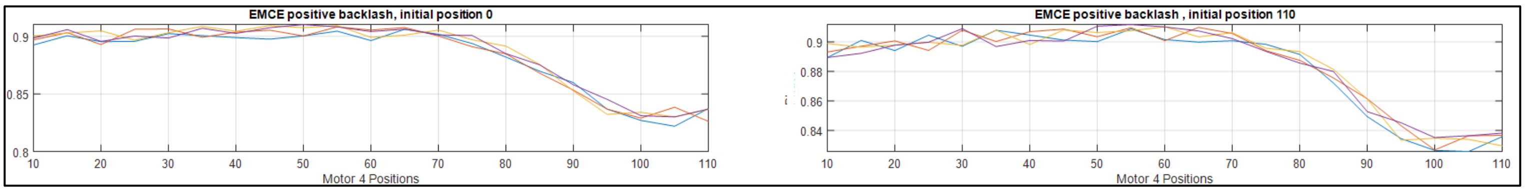


Figure 51 – Magnitude of S_{11} after update

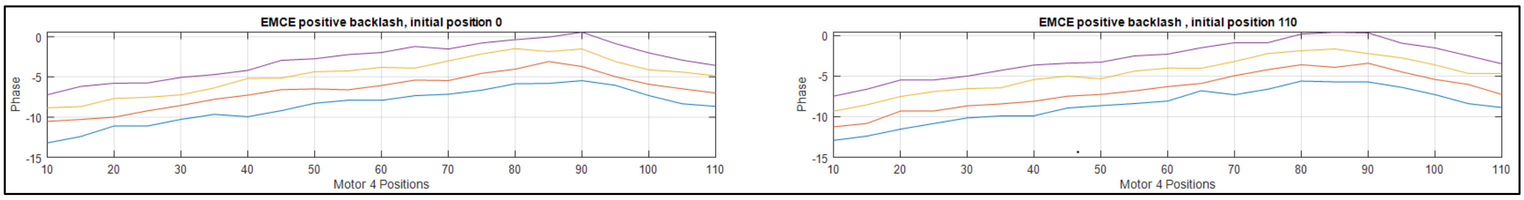


Figure 52 - Phase of S_{11} after update

From Figure 50 and 51 it could be seen that S_{11} parameter was no more initial position dependent. In other words, no matter the initial position was or the intermediate steps were, the tuner controller was able to provide the same RF characteristics for the same motor position and same motor configuration. This was the most crucial step in terms of maintaining reliable, reproducible and stable RF performances.

It was a huge surprise that the old Maury tuner controller was not able to provide consistent RF characteristics for same motor positions. After the mentioned experiments and measurements, this problem was fixed and implemented to the new tuner controller in a fixed way.

All these processes were repeated by moving different motors, with different initial and final and with different configurations. But for the ease of explanation, only one case was explained and demonstrated.

9. CONCLUSION

During my internship, I was tasked with upgrading an old RF impedance tuner controller by replacing it with a more advanced model that included new features. The project encompassed a variety of tasks, including cable soldering, basic signal processing, automated testing, and RF measurements. I was responsible for each step in the development process, which allowed me to gain hands-on experience across multiple techniques. I had the opportunity to work with a range of instruments and components, such as the TRINAMIC motor controller board, VNA, and stepper motors.

This experience not only helped me to advance my problem-solving skills but also taught me how to approach technical challenges systematically. Ultimately, we developed a functioning MATLAB-controlled RF impedance tuner controller.

I had the privilege of working under the guidance of Mr. Arthaber, whose support and expertise were invaluable throughout the project. I am also grateful to the entire team at EMCE for their assistance and collaboration. It was a rewarding and enriching experience to be part of such a skilled research team.

10.REFERENCES & DATASHEETS

Agilent PNA: https://www.testworld.com/wp-content/uploads/user-guide-help-agilent-e8362b-e8363b-e8364b-e8361a-n5230a-n5242a-pna-series-microwave-network-analyzers.pdf

TRINAMIC Motor Controller Board: https://www.analog.com/media/en/technical-documentation/user-guides/TMCM-6212_hardware_manual_hw1.10_rev1.01.pdf

TMCL-IDE: https://www.analog.com/media/en/dsp-documentation/software-manuals/TMCM-6212_TMCL_firmware_manual_Fw1.11_Rev1.07.pdf

Motor PX43-04AA:
https://www.jameco.com/Jameco/Products/ProdDS/2270927.pdf?srsltid=AfmBOoowz1cxCA9gf8rKzIRSxdpPqzxTriJmChZJvo0vlyjFWBzrU6DJ

Motor Phytron: https://www.microprivod.ru/assets/files/pdf/catalogue/phytron/motors/zss-en.pdf