Development of a portable, FPGA-based metal detector for humanitarian demining

Laboratorium for Solid State Physics
Deparment of Physics
ETH Zurich

by
Yiming Alan Li

Supervisor
Dr. Yves Acremann
Joël Rehmann

in the group of
Prof. Dr. Andreas Vaterlaus

14.03.2024

Abstract

This semester work presents the results of a metal detector capable of differentiating between various sizes of different metals using Magnetic Field Induction Spectroscopy. Through designing the device in a portable fashion, the main purpose is to detect unexploded ordnance burried beneath the surface. By analyzing the response in the frequency domain, we get a characteristic spectrum of different metals. This thesis presents the problems encountered and the experimental results of the development of this metal detector.

Contents

1 Introduction **3**

2 Theory **4**
2.1 Electromagnetic Induction Spectroscopy . . . 4
2.2 Eddy currents . . . 4
2.3 Magnetisation . . . 4
2.4 Skin Effect . . . 4

3 Design **6**
3.1 Necessary electrical components . . . 6
3.1.1 Flux gate sensor . . . 6
3.1.2 FPGA . . . 6
3.2 Board . . . 7
3.3 mechanical components and design . . . 8
3.4 Determining the coil parameters . . . 8

4 Experimental setup **10**
4.1 Procedure . . . 10
4.2 Examined metal pieces . . . 10
4.3 Code . . . 11

5 Results **12**
5.1 Dependency on size . . . 12
5.1.1 Copper ring . . . 12
5.1.2 Stainless steel . . . 12
5.2 Dependency on distance . . . 13
5.2.1 Copper ring . . . 13
5.2.2 Stainless steel . . . 14
5.3 Amplitude-distance dependence . . . 16
5.3.1 Amplitude at minimum against distance . . . 17
5.3.2 Average amplitude against distance . . . 18

6 Discussion **19**

7 Conclusion **20**

1 Introduction

There exist different types of landmines intended to either kill humans (antipersonnel mines) or destroy vehicles and they are designed in a way to be triggered by victims, e.g. when pressure is applied from the top. If not triggered, these landmines can stay active for many decades, thereby posing a significant risk to people and environments and are still today the cause of many deaths of innocent lives, most of whom are children living in previous war-torn countries. The clearing of unexploded ordnance (UXO) remains a challenging task with nowadays methods being too slow and often dangerous for the people involved. Most notable methods include mine sniffing dogs, ground penetrating radar, mechanical clearing and metal detectors.
The advantage of using a metal detector for humanitarian demining is the expected low false negative rate since UXOs contain metal. This low false negative rate is crucial and saves lives. The disadvantage is the high false positive rate. These false alarms happen most of the time due to other metal objects present in the soil such as bullets, grenade pins or even everyday items such as a soda can.
From this, it becomes clear that we need to develop a portable metal detector which can differentiate between different sizes of different metals. To achieve this, we apply a time varying current to an excitation coil which generates a time varying magnetic field. Two sensitive flux gate sensors positioned on the same axis measure the magnetic field generated by the metal. A Field Programmable Gate Array (FPGA) is responsible for the continuous data acquisition and processing. The following chapters give an explanation of the underlying physics and the design of the device as well as the experiments performed with their results.

2 Theory

2.1 Electromagnetic Induction Spectroscopy

Electromagnetic Induction Spectroscopy (EMIS) is a method used to determine electromagnetic properties of materials. By applying a changing magnetic field, eddy currents will be induced in the metal which will then in turn create a secondary magnetic field. It is in this secondary magnetic field that one can find information about the metals different electrical and magnetic properties. By analyzing the response in the frequency domain, one hope to be able to uniquely identify the object buried under the surface. MIS being a contactless and non-destructive technique makes it particularly suitable for humanitarian demining where one certainly does not want to alter or even trigger the explosives.

2.2 Eddy currents

Eddy currents arise due to Faraday's Law in Electrodynamics. A changing magnetic field induces a current around the loop of a conducting material. This is also reflected in the third Maxwell's equations:

$$\nabla \times \mathbf{E} = \frac{\partial \mathbf{B}}{\partial t} \tag{1}$$

or written in integral form:

$$\oint_{\partial A} \mathbf{E} \cdot \mathbf{ds} = -\int_{A} \frac{\partial \mathbf{B}}{\partial t} \cdot \mathbf{dA} \tag{2}$$

with the electric field $\mathbf{E}$, the magnetic field $\mathbf{B}$ and the area of integration $\mathbf{dA}$.
Since the current resulting from $\mathbf{E}$ is dependent on $\frac{\partial \mathbf{B}}{\partial t}$, the response expressed in Fourier components would be shifted by 90° resulting in a purely imaginary part. It is also worth noting that different electrical conductivity and different sizes give space for more current to flow. Therefore the theory predicts a different response for different sizes and material.

2.3 Magnetisation

Ferromagnetic materials experience magnetisation due to the applied magnetic field. The magnetisation is given through

$$\mathbf{M} \approx \chi \mathbf{H} = \frac{\chi_m}{\mu_0 \mu_r} \mathbf{B} \tag{3}$$

with magnetisation $\mathbf{M}$, the magnetic field $\mathbf{B}$, the magnetic susceptibility χ_m, the free permeability μ_0 and the relative permeability μ_r. The response due to magnetisation expressed in Fourier components would only have a real part since it is directly proportional to $\mathbf{B}$.

2.4 Skin Effect

The skin effect describes a tendency of Eddy currents induced by an alternating magnetic field to flow mostly on the surface.The current density decreases exponentially with greater depths in the conductor. The so called skin depth is the depth at which the current density is equal to $1/e$ of its value at the surface.
One can calculate the skin depth δ with Maxwell's equations

$$\delta = \sqrt{\frac{2\rho}{\omega \mu}} \tag{4}$$

where ρ the resistivity of the material, ω the angular frequency of the alternating current and μ the permeability of the conductor. This effect also indicates a difference of Eddy currents for different sizes of metal therefore giving another argument of why it should be possible to distinguish between different sizes.

3 Design

3.1 Necessary electrical components

3.1.1 Flux gate sensor

A fluxgate sensor is a ring core of a magnetically permeable alloy with two coils wrapped around it. The first coil around the ring is called the drive winding and is fed by a signal in the form of a sinusoidal wave. This will produce a magnetic field inside the core. This second coil which is wrapped around both the core and the drive coil is called the sense winding. When there is no external magnetic field applied, there will be no net change of flux in the sense winding and therefore no voltage induced.
However, if an external field is present, the two halves of the core will have a magnetic field in opposite directions leading to a different saturation time. The magnetic field will not cancel out the sense winding therefore measures a change in flux. Through measuring the voltage induced by the change in flux, one can determine the strength of the magnetic field.

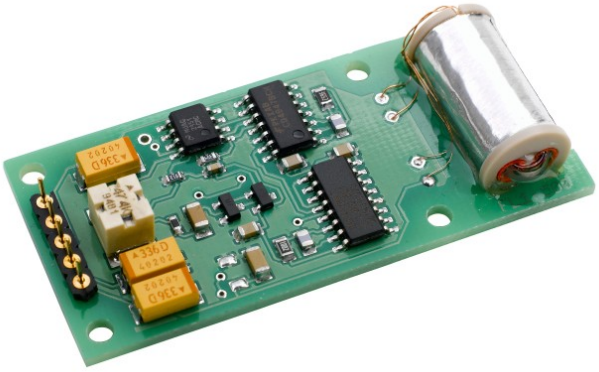


Figure 1: Fluxgate Sensor FL1-500 from Stefan Meyer Instruments

3.1.2 FPGA

For the signal processing part, a FPGA is necessary to ensure fast data acquisition and data processing. The FPGA used in this project was the **Cora Z7-07S** by Digilent Inc. It combines an FPGA as well as an ARM-based processor. We can program the necessary hardware on the FPGA and run PYNQ, an open source project from Xilinx, on the ARM-processor to interface with it through a Jupyter Notebook.

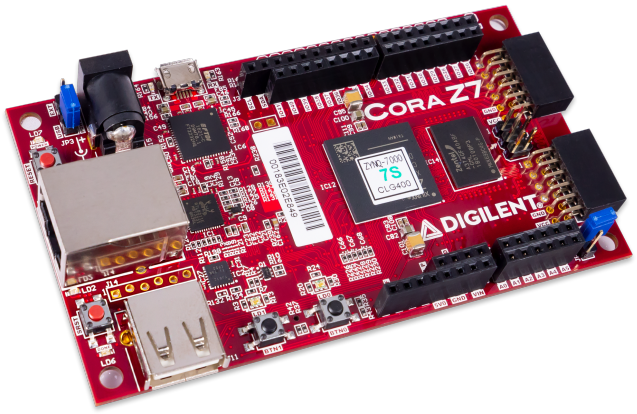


Figure 2: FPGA development board Cora Z7-07S

3.2 Board

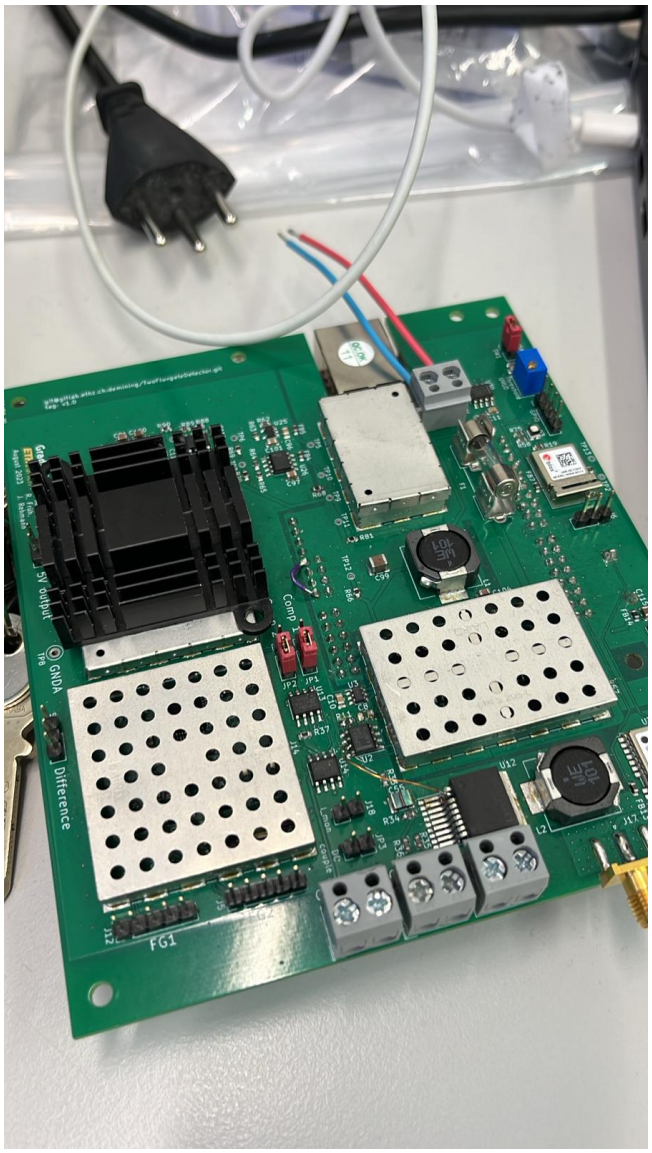
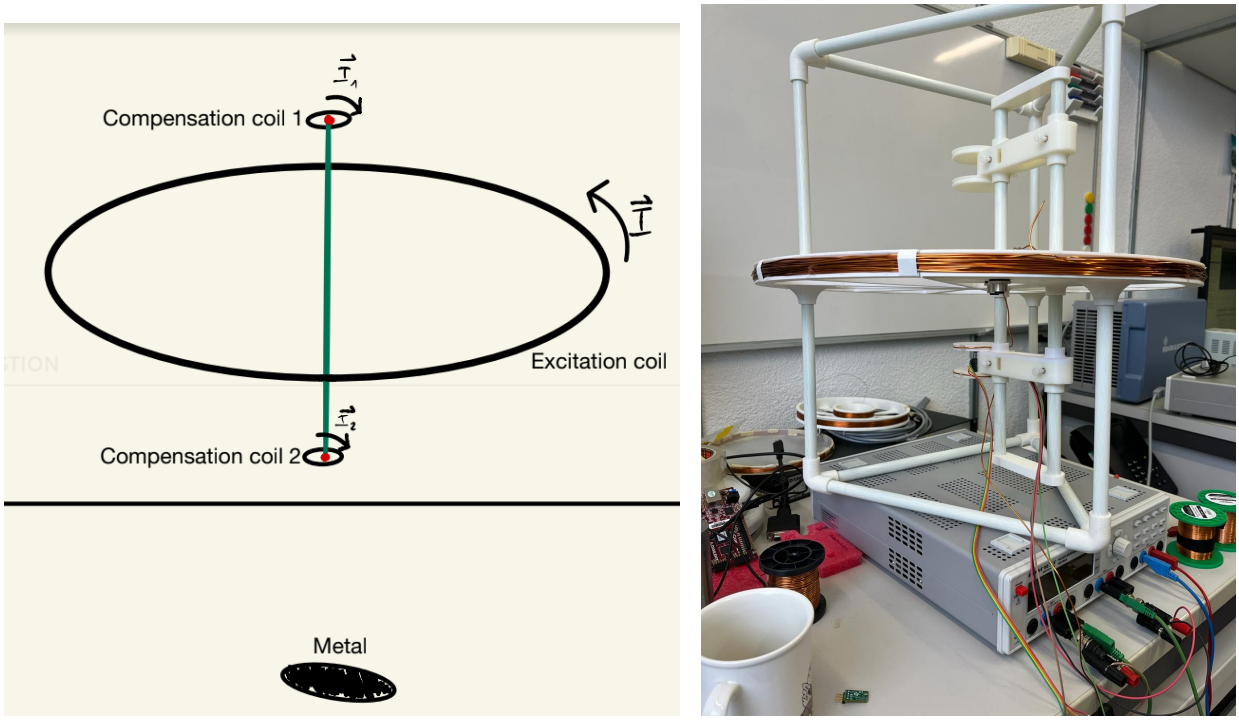


Figure 3: The board developed for this project

The above figure shows the board that was used in the project. It includes all necessary electrical components to drive two flux gate sensors, two compensation coils as well as one drive coil. It also integrates a Bluetooth module and a RTK-capable GPS receiver. The schematics and the board design was developed by Robert Früh, Dr. Yves Acremann and Joel Rehmann.

3.3 mechanical components and design



(a) simplified design on paper

(b) Actual design

Figure (a) shows a simplified version of the design on paper. Figure (b) shows the actual design. It consists of an excitation coil with a radius of 25cm and two flux gate sensors. Around these two flux gate sensors there is a compensation coil to compensate the magnetic field generated by the excitation coil. The goal is to read a zero magnetic field at these positions when there is no metal around. Once there is metal nearby, the sensors will only measure the secondary magnetic field induced by the metal. It is possible to compensate most of the magnetic field of the excitation coil by winding the same wire of the excitation coil and therefore letting the same current run through the compensation coil but in opposite directions. By doing this, we not only escape the problem of determining the right transfer function for the compensation current but also eliminating the noise of the compensation current which now is the same as the one from the excitation current.

3.4 Determining the coil parameters

There are two goals when determining the right choice of coil parameters for the device. The first one is obviously to create a strong magnetic field with the excitation coil. The second one is to choose the them in a way so that it is possible to wind them around the compensation coil using the same wire but in opposite direction. The number of turns around the excitation coil and compensation coils needs to be an integer. The wire we will choose in the end is of course

also dependent on what is available on the market.
The excitation coil has a radius of $r = 25\text{cm}$ and a length of $a = 2.1\text{cm}$. We will work with a voltage of around 15V and we can set the frequency to around $f = 100\text{Hz}$. The current through the excitation coil is given by

$$I = \frac{U}{|Z|} \tag{5}$$

with the complex impedance Z given by

$$Z = \underbrace{R}_{\text{Resistance}} + \quad i \underbrace{\omega L}_{\text{Reactance}} \tag{6}$$

The resistance R is given by

$$R = \rho \frac{L}{A} = \rho \frac{N \cdot 2\pi r}{\pi(\frac{d}{2})^2} \tag{7}$$

with the resistivity ρ, the total length of the wire L, the cross section of the wire A, the number of turns around the excitation coil N and the diameter of the wire d.
Furthermore, the number of turns N are related to d through

$$N = \frac{a}{d} \tag{8}$$

The inductance is given by

$$L = \mu_0 \frac{N^2 \cdot A}{a} = \mu_0 \frac{a \cdot \pi r^2}{d^2} \tag{9}$$

with the permeability of free space μ_0 and the area of the coil A. Note that this formula usually only applies to solenoid coils with a length much bigger than the diameter.
Furthermore, we have for copper

$$\rho = 1.72 * 10^{-8} \Omega m \qquad \mu_0 = 4\pi 10^{-7} N/A^2 \tag{10}$$

With these formula one can now calculate the current I.
The magnetic field outside the coil can be calculated by stacking the magnetic field of N rings where N is the number of turns. We have

$$B_{ec} = N * \frac{\mu}{2} I \frac{r^2}{(r^2 + z^2)^{\frac{3}{2}}} \tag{11}$$

The two compensation coils are Helmholtz coils. Since the flux gate sensor is positioned at the center of the Helmholtz coil we need to give an expression for the number of turns with respect to the current flowing and the magnetic field at the center. This is given through

$$B_{HH} = (\frac{4}{5})^{\frac{3}{2}} \mu_0 \frac{N_{HH} \cdot I}{R} \Rightarrow N_{HH} = \frac{B_{HH}}{(\frac{4}{5})^{\frac{3}{2}}} \cdot \frac{R}{\mu_0 \cdot I} \tag{12}$$

with the radius $R = 3\text{cm}$ of the Helmholtz coil. As already mentioned, N needs to be an integer. Since B_{HH} needs to compensate B_{ec}, it turns out that for a distance of $z \approx 11.5\text{cm}$ we need exactly one turn. The code for these calculations can be found in Appendix A

4 Experimental setup

4.1 Procedure

The experiments are taking place in the laboratory. The device and the metal pieces are placed on top of a box where the metal pieces are at a distance of 12.5cm away from the nearest flux gate sensor. After that, the different metal pieces are placed at a certain distance away from the sensors on top of wood blocks. By removing the wood blocks one by one and measuring the response, we hope to get data which gives us insights into the distance dependence of the response signal. This experiment is repeated until there are no more wood blocks left, placing the metal at a distance of around 50cm away from the sensor.

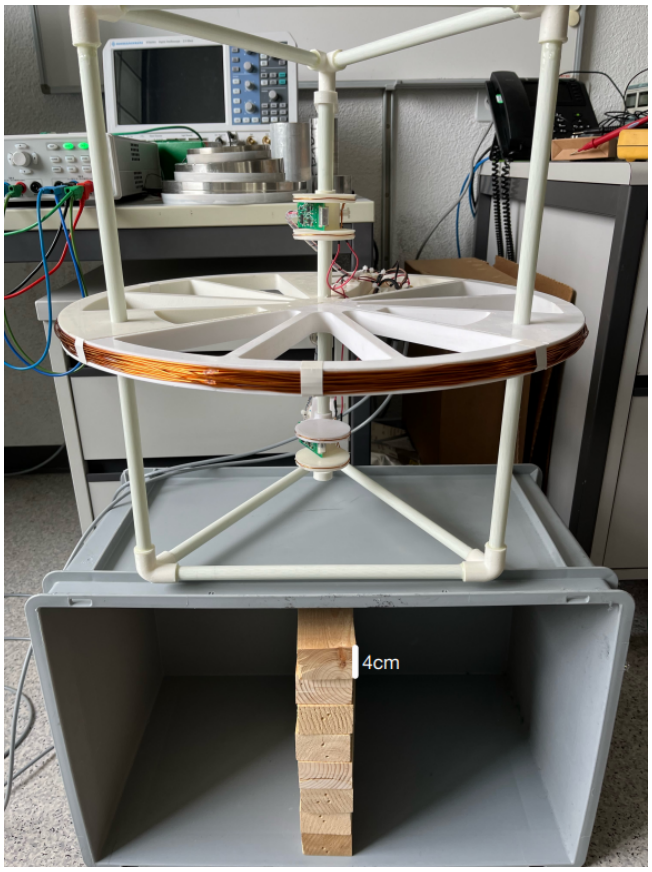


Figure 5: Experimental setup in the laboratory. The wood blocks have a height of (4 ± 0.1) cm By placing the metal pieces on top of the wood blocks and removing them one by one, we try to investigate the distance dependence of the response signal. The distance is measured with respect to the flux gate sensor closest to the metal pieces.

4.2 Examined metal pieces

Figure 2 shows the examined pieces of metal which were available in the laboratory. Copper is known for its excellent conducting properties. We therefore expect the Eddy currents to flow without a problem. There are four different copper rings available with a radius of 4cm, 6.05cm, 8.55cm and 13.75cm.

(a) copper rings (b) stainless steel discs

Figure 6: Comparison of two different metals and various sizes. For the case of stainless steel

For the case of stainless steel, we have disks of radius 5.7cm, 7.6cm and 10.1 cm which we will denote as small stainless steel, middle stainless steel and big stainless steel, respectively.

4.3 Code

The code for the data acquisition during the experiment can be found in Appendix B . The code for the data processing and comparing can be found in Appendix C.

5 Results

The following data is obtained after performing 300 measurements for each material for each distance and averaging the results. The procedure is explained above. The resolution of the frequency domain is given by $\delta f = \frac{48000}{2048} = 23.4375\text{Hz}$. Furthermore, a Savitzky-Golay Filter with window length 11 and polyorder 3 was applied to all the averaged data.

5.1 Dependency on size

5.1.1 Copper ring

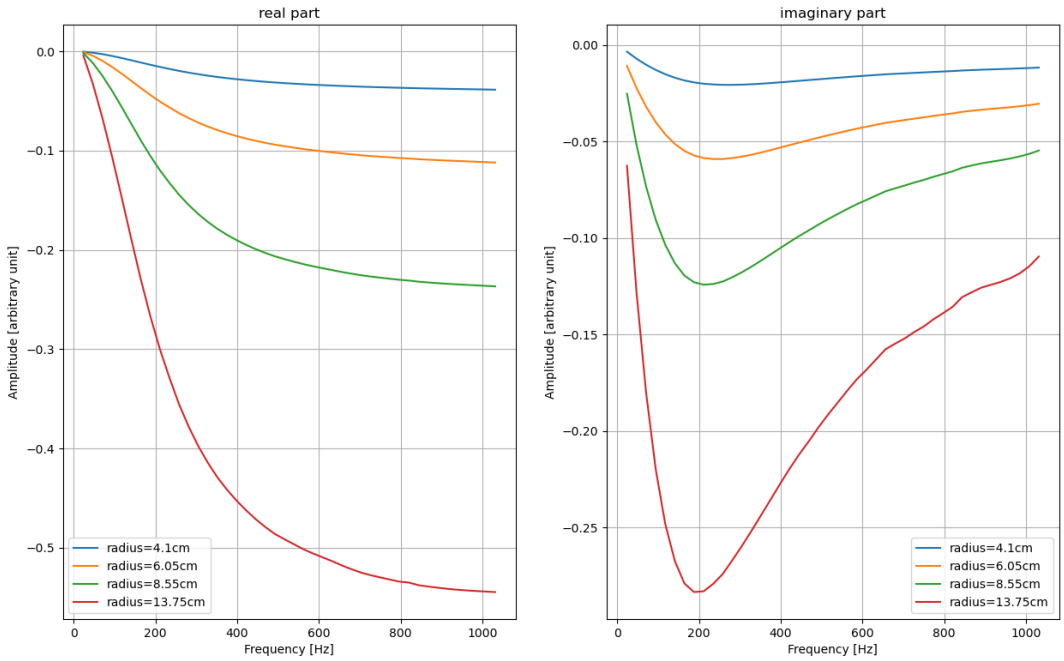


Figure 7: Comparison of different sizes for a copper ring. Left: real part; Right: imaginary part
The measurements were taken at a distance of 12.5cm to the sensor

The characteristic spectrum for copper ring can be seen especially in the imaginary part where the minimum is at a frequency of $f \approx 281.3\text{Hz}$, $f \approx 257.8\text{Hz}$, $f \approx 210.9\text{Hz}$ and $f \approx 187.5\text{Hz}$ for the radii 4.1cm, 6.05cm, 8.55cm and 13.75cm respectively.

5.1.2 Stainless steel

After following the procedure for stainless steel we get the following data. Stainless steel small, middle and big are abbreviated by sss, ssm and ssb respectively.

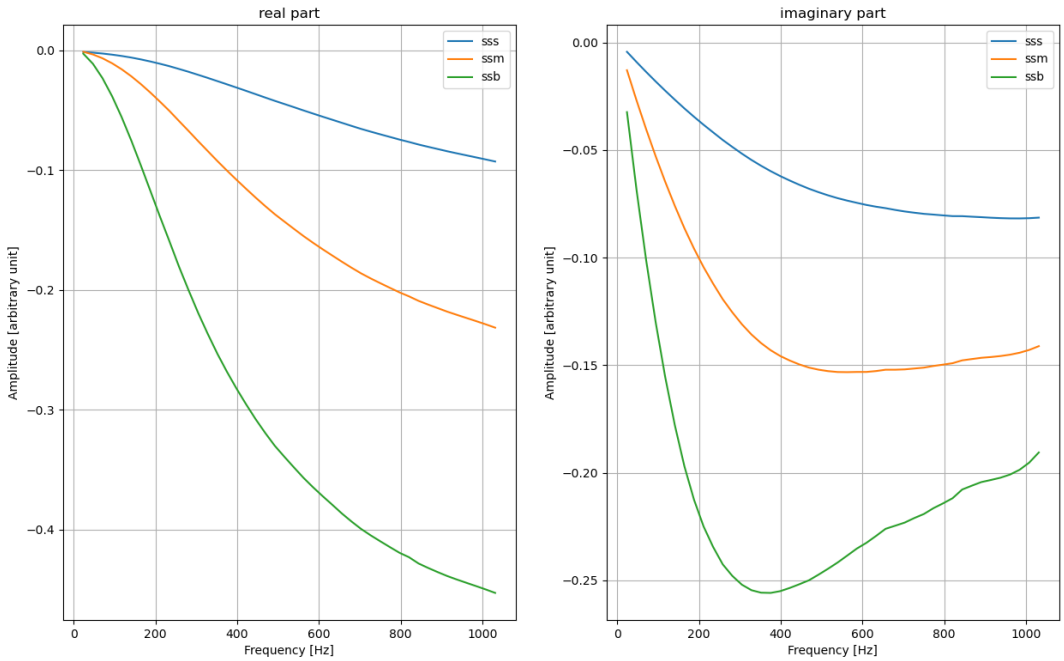


Figure 8: Comparison of different sizes for stainless steel. Left: real part; Right: imaginary part
The measurements were taken at a distance of 12.5cm to the sensor

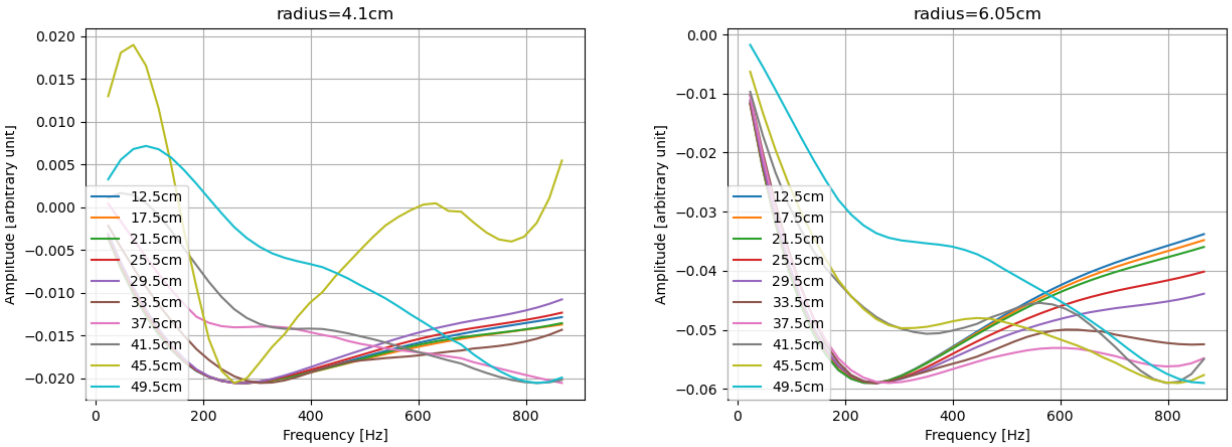
The characteristic spectrum for stainless steel can be seen especially in the imaginary part where the minimum is at a frequency of $f \approx 984.4$Hz, $f \approx 562.5$Hz, $f \approx 375.0$Hz for the sizes small, middle and big respectively.

5.2 Dependency on distance

In the last section we successfully could differentiate between different sizes of the same material at the same distance by looking at the spectrum. Especially the minimum frequency was different. However, it could be that this minimum frequency shifts by changing the distance. If that is the case, one could neither get information about the size nor about the material if that shift is significant. More measurements at different distances had to be taken to investigate this further. The amplitude of the response obviously gets weaker with increasing distances, therefore, the curves need to be scaled to the one representing the smallest distance.

5.2.1 Copper ring

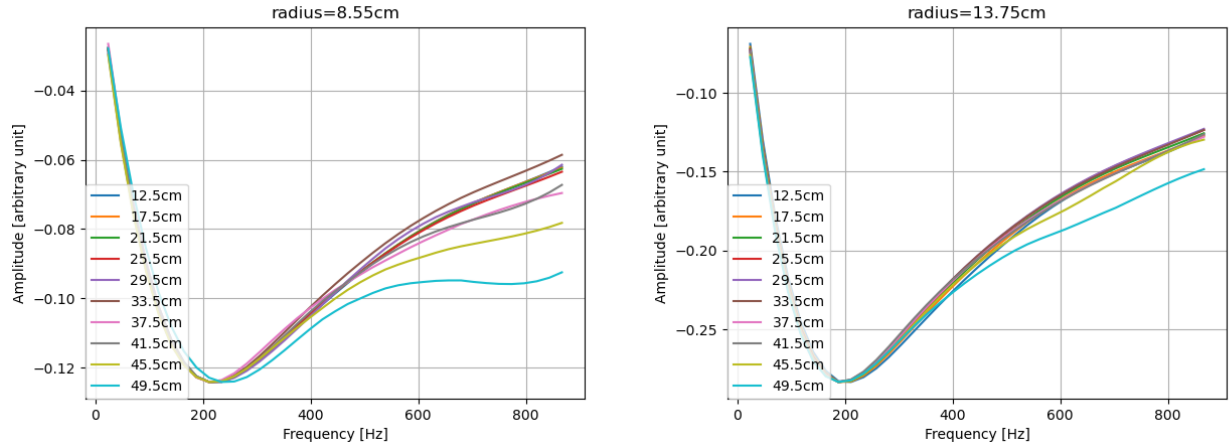
Ten measurements were done and the curves were scaled on to each other.



(a) copper ring with radius 4.1cm (b) copper ring with radius 6.05cm

Figure 9: Comparison of smaller copper rings

We can clearly see that for distances smaller than around 37.5cm the curves all overlap and only start to deviate for higher frequencies. This is a good sign telling us that this spectrum is truly unique and shows the same behaviour independent of distance in this distance range. For distances larger than 37.5cm the noise takes over and the curves do not overlap anymore making it very hard to make sense of the data.



(a) copper ring with radius 8.55cm (b) copper ring with radius 13.75cm

Figure 10: Comparison of bigger copper rings

For bigger sizes and especially for the copper ring with radius 13.75cm, the curves all overlap for lower frequencies making this spectrum unique. However, we can see from the left figure that there is tendency for the frequency minimum to shift towards higher values for bigger distances. For frequencies above 500Hz we see that deviations start to occur.

5.2.2 Stainless steel

Following the same procedure like we did with the copper rings we get for stainless steel

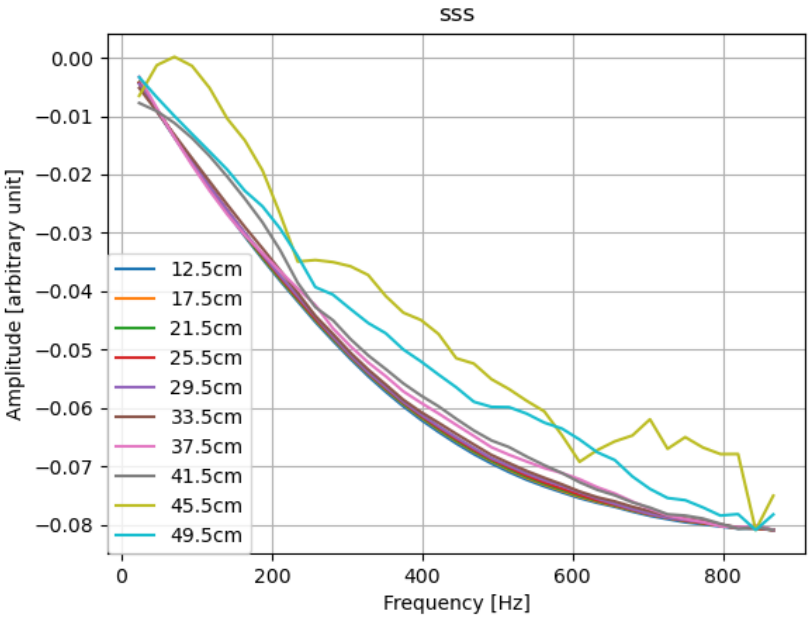


Figure 11: response signal for stainless steel small for different distances scaled on to each other

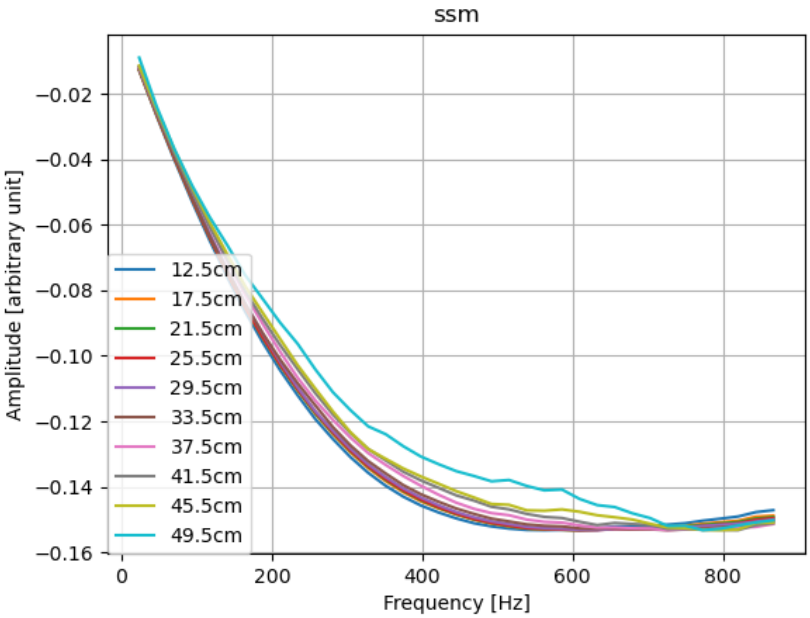


Figure 12: response signal for stainless steel middle for different distances scaled on to each other

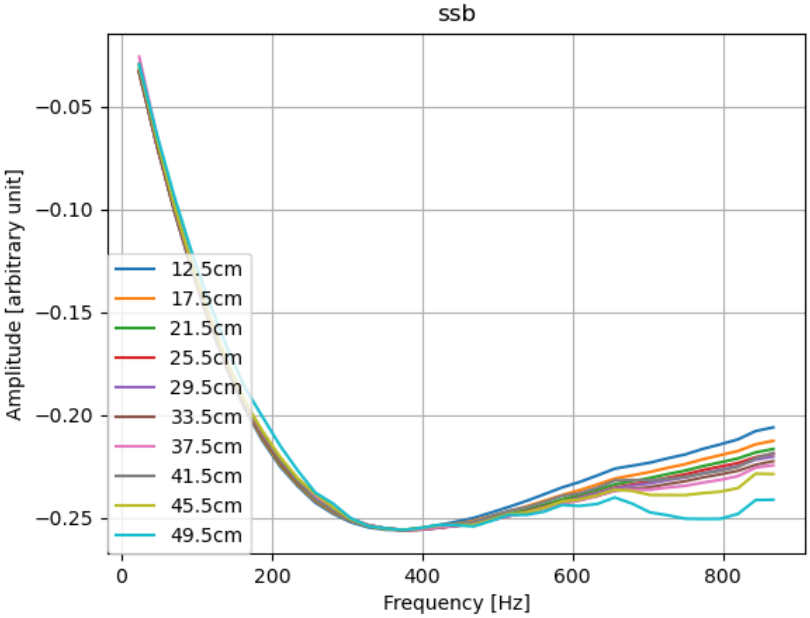


Figure 13: response signal for stainless steel big for different distances scaled on to each other

5.3 Amplitude-distance dependence

Last but not least, we try to investigate the other axis of the spectrum namely the amplitude-distance relationship. It is worth looking into two different amplitudes, namely the amplitude at the minimum of the imaginary part of the spectrum. The other is the amplitude which is calculated by taking the average over all the amplitudes of every frequency component in both the real and imaginary part. The absolute values of these amplitudes are plotted against distance in a double logarithmic plot to determine the underlying power law.

5.3.1 Amplitude at minimum against distance

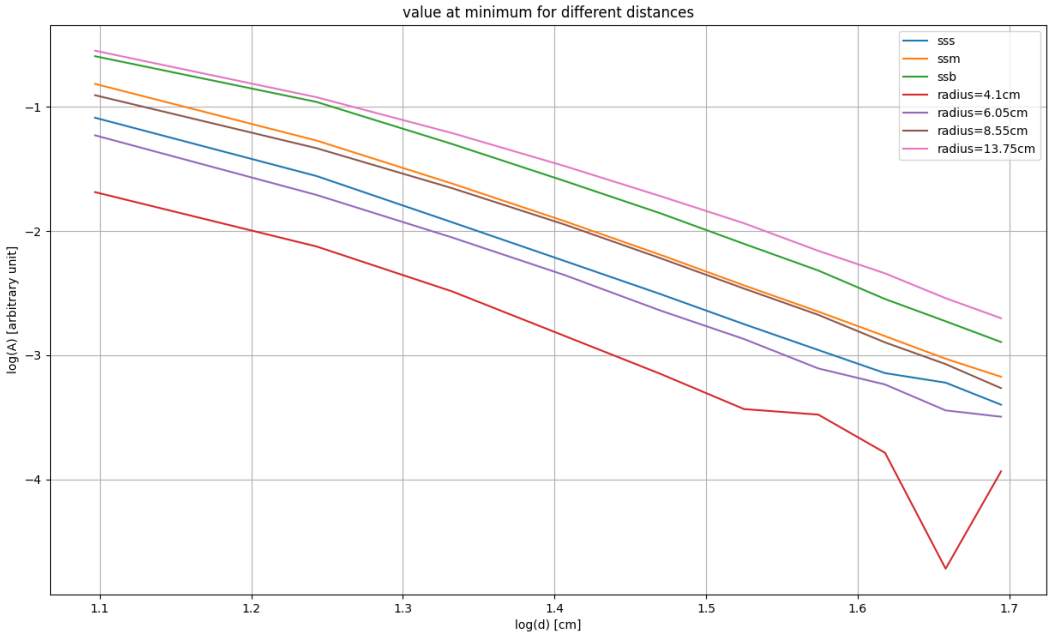


Figure 14: amplitude at minimum in imaginary part plotted against distance on a double logarithmic scale

The following table summarizes the slopes of the curves in this figure. The data was fitted using the scipy.stats.linregress function

	sss	ssm	ssb	c-ring 4.1cm	c-ring 6.05cm	c-ring 8.55cm	c-ring 13.75cm
$\frac{dA}{dz}$ $[\frac{*}{cm}]$	-3.97±0.07	-4.05±0.09	-3.98±0.14	-4.50±0.48	-3.96 ±0.08	-4.02 ±0.13	-3.68 ±0.14

Table 1: Caption

5.3.2 Average amplitude against distance

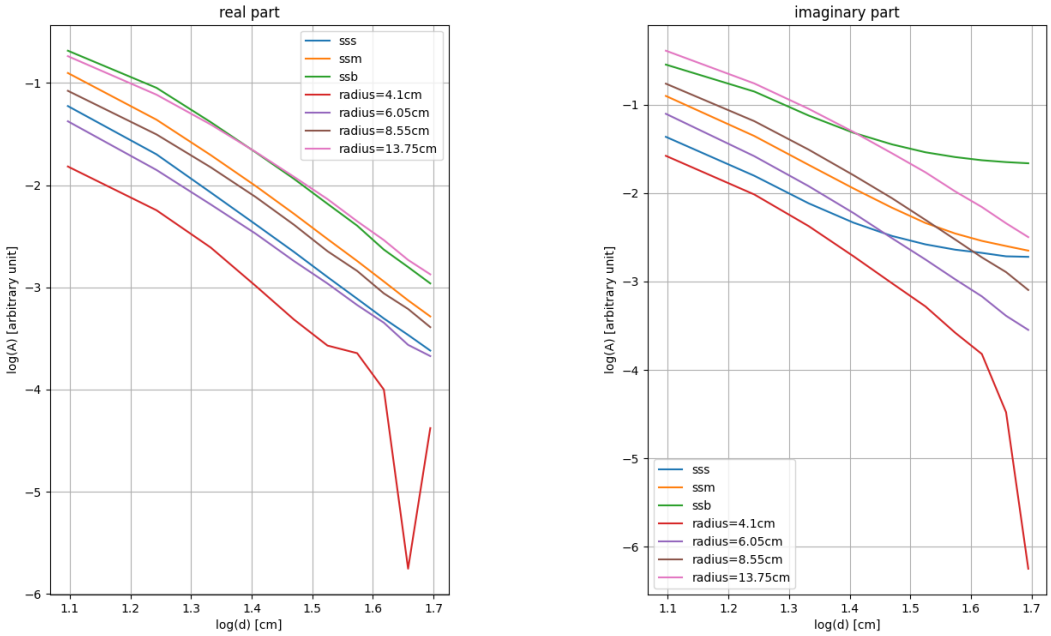


Figure 15: Average amplitude plotted against distance on a double logarithmic scale, Left: Real part; Right: Imaginary part

One can see from the figures that the average amplitudes follow a straight line for the most part. The following table summarizes the slopes of the curves in these figures. The data was fitted using the scipy.stats.linregress function

	sss	ssm	ssb	c-ring 4.1cm	c-ring 6.05cm	c-ring 8.55cm	c-ring 13.75cm
$\frac{dA_{real}}{dz}$ $[\frac{*}{cm}]$	-4.10±0.07	-4.08±0.09	-3.95±0.14	-5.31±0.90	-3.93 ±0.07	-3.96 ±0.11	-3.67 ±0.13
$\frac{dA_{imag}}{dz}$ $[\frac{*}{cm}]$	-2.30±0.19	-3.03±0.12	-1.94 ±0.15	-6.24 ±1.11	-4.16 ±0.10	-3.96 ±0.12	-3.61 ±0.12

Table 2: Caption

6 Discussion

The theory behind Magnetic Field Induction Spectroscopy predicts that Eddy currents manifest itself in the imaginary part of the response signal in the frequency domain while magnetization is directly proportional to the real part. The experimental results showed distinct curves for all the examined metal pieces especially a characteristic minimum in the imaginary part thereby making the behaviour of each metal truly unique. After varying the distance for each material this trend keeps to follow until eventually we can see a shift in the higher frequencies above 500Hz which we yet cannot explain. This shift is stronger for larger distances and smaller metal pieces. However, for our two biggest metal pieces namely the bigger stainless steel and the copper ring with a radius of 13.75cm the minima frequency stays the same even for a distance of half a meter. Since metals of such sizes are expected to be present in the unexploded ordnance we can convincingly claim to be able to detect those.
Furthermore, the real part of the response signal can be used to distinguish between ferromagnetic and less ferromagnetic metals and a further investigation could be beneficial.
When trying to determine the actual depth in which the metal piece is located under the surface, a first step was to analyze the dependence of the amplitudes with the distance. When plotted against each other in a double logarithmic scale, the slope tends to be around 4.0 for almost all metal pieces in both the real part as well as the imaginary part regardless of whether we look at the amplitude averaged over the whole spectrum or the amplitude at this distinct minimum frequency in the imaginary part. This suggests a $\frac{1}{z^4}$ dependency with z being the distance perpendicular to the surface. To now determine the actual depth, the gradient should provide us with enough information because it measures the difference in the magnetic field at two points at a fixed distance to each other. A more convincing model with a theoretical explanation of this distance relationship is desirable.

7 Conclusion

During this semester project, the goal was set to design a device capable of detecting metal and differentiating between different sizes through a method known as Magnetic Field Induction Spectroscopy. Further requirements such as the portability led to design constraints. When testing with the examined metal pieces which consist of copper rings in four different radii as well as stainless steel in three different radii, the results showed a characteristic behaviour in the imaginary part of the response signal when looked at in the frequency spectrum therefore making this a successful first prototype.
To actually compute the distance we need to look at the gradient of the magnetic field. Once this is achieved, another idea for future prototypes would be to extend the overall design to include three excitation coils and three gradient sensors for each axis. There exist literature where one can compute with these information the Magnetic Polarizability Tensor and diagonalize it to get the relative orientation of the metal piece under the surface.
Including GPS and Bluetooth applications to the device would further enhance the portability of the device.
Another experiment which involves placing smaller metal debris above the bigger metal pieces or possibly a real type of ordnance would be the next step to see whether this first prototype can live up to its purpose under real conditions.

References

[1] Claudio Bruschini, *A multidisciplinary analysis of frequency domain metal detectors for humanitarian demining*, Vrije Universiteit Brussel, Faculty of Applied Sciences, Departement of Electronics and Information Processing

[2] Toykan Özdeger, *Advances in Techniques for the characterisation of targets in metal detection and ultrawide band electromagnetic screening applications*, University of Manchester, Department of electrical and electronic engineering

Appendix

Appendix A Determining the coil parameters

```
import numpy as np
import matplotlib.pyplot as plt

U = 15 #[V]
#I = 5 #[A]
f = 100 #[Hz]
r = 0.25 #[m]
a = 0.021 #[m]   #length of the solenoid
#d = 0.5 #[mm]



#constants
pi = 3.14
mu = 4*pi*(10**-7)                                  #N/(A^2)
rho = 1.72 * 10**-8 #1/sig                  #resistivity [Ome * m]


d = 1.32
d = d/1000 ## mm -> m


N = a/d
print("Number of turns:          ",N)
print(" ")


def calculateRea():
        A_sol = pi*r**2                                         #cross
           ↪ section of the coil
        l = 2*pi*r                                                      #
           ↪ average length of one coil
        L = np.power(N,2) * mu * A_sol/a #inductance
        w = 2*pi*f
        Rea = w*L
           ↪ #reactance
        print("Reactance:        ", Rea)
        print(" ")
        return Rea

def calculateRes():
        A_wire = pi * np.power(d/2, 2)  #cross section of wire
        l_tot = N*(2*pi*r)                                      #total
           ↪ length of the wire
```

```
41          Res = rho*l_tot/A_wire                                  #
              ↪ resistivity
42          print("Resistance:       ", Res)
43          print(" ")
44          return Res
45
46
47  def calculateZ(Rea, Res):
48          Z = np.sqrt(np.power(Res,2) + np.power(Rea,2))
49          print("Z:          ", Z)
50          print(" ")
51          return Z
52
53  def calculateI(Z):
54          I = U/Z
55          print("I:          ", I)
56          print(" ")
57          return I
58
59  def calculateB_inside(I):
60          B = mu*I*N/a
61          print("Magnetic field B inside the coil:          ", B)
62          print(" ")
63          return B
64
65  def calculateB_outside(I, z): #calculate the magnetic field
      ↪ outside on z axis
66          B = mu*N*r**2*I/(2*np.power(r**2+z**2, 3/2))
67          print("Magnetic field B outside the coil:          ", B)
68          print(" ")
69          return B
70
71  Rea = calculateRea()
72  Res = calculateRes()
73  Z = calculateZ(Rea, Res)
74  I = calculateI(Z)
75  B = calculateB_outside(I, 0.115)
76
77
78  ############### calculing the parameters of the Helmholtz-
      ↪ compensation solenoid
79  c = np.power(4/5,3/2)
80  R = 0.03
81
82  def calculate_N(B, I):
83          N = B/c * (R/(mu * I))
84          print("number of turns needed:  ",N)
85          print(" ")
```

```
        return N

N = calculate_N(B,I)
```

Appendix B Data Acquisition

```
# importing all neccessary modules
import pynq
from pynq import Overlay
import asyncio
from pynq import Interrupt
from pynq import MMIO
import time
import matplotlib.pyplot as plt
import numpy as np
import copy
import math
from numpy.fft import fft, ifft
import random
import codecs, json
```

```
ol = Overlay("./Overlays/overlay1.bit")
```

```
#BRAM
bram_in_L         = MMIO(0x40000000, 0x4000)
bram_in_R         = MMIO(0x42000000, 0x4000)
bram_out_L        = MMIO(0x44000000, 0x4000)
bram_out_R        = MMIO(0x46000000, 0x4000)
bram_exc          = MMIO(0x48000000, 0x2000)
bram_gradient     = MMIO(0x4A000000, 0x4000)
bram_exc_current  = MMIO(0x4C000000, 0x4000)

bram_in_reg_L_0 = bram_in_L.array[0:2048]
bram_in_reg_L_1 = bram_in_L.array[2048:4096]
bram_in_reg_L_0.dtype = np.int32
bram_in_reg_L_1.dtype = np.int32

bram_in_reg_R_0 = bram_in_R.array[0:2048]
bram_in_reg_R_1 = bram_in_R.array[2048:4096]
bram_in_reg_R_0.dtype = np.int32
bram_in_reg_R_1.dtype = np.int32

bram_out_reg_L_0 = bram_out_L.array[0:2048]
bram_out_reg_L_1 = bram_out_L.array[2048:4096]
bram_out_reg_L_0.dtype = np.int32
bram_out_reg_L_1.dtype = np.int32

bram_out_reg_R_0 = bram_out_R.array[0:2048]
```

```
bram_out_reg_R_1 = bram_out_R.array[2048:4096]
bram_out_reg_R_0.dtype = np.int32
bram_out_reg_R_1.dtype = np.int32

bram_exc_reg = bram_exc.array
bram_exc_reg.dtype = np.int32

bram_gradient_reg_0 = bram_gradient.array[0:2048]
bram_gradient_reg_1 = bram_gradient.array[2048:4096]
bram_gradient_reg_0.dtype = np.int32
bram_gradient_reg_1.dtype = np.int32

bram_exc_current_reg_0 = bram_exc_current.array[0:2048]
bram_exc_current_reg_1 = bram_exc_current.array[2048:4096]
bram_exc_current_reg_0.dtype = np.int32
bram_exc_current_reg_1.dtype = np.int32
```

```
#Interrupts
ol.interrupt_pins
interrupt_0 = Interrupt('Memory_controller/interrupt_0')
interrupt_1 = Interrupt('Memory_controller/interrupt_1')
```

```
#Define ISR
gradient_data_in = np.full(2048, 0)
exc_current_data_in = np.full(2048, 0)
sensor_data_in_1 = np.full(2048, 0)
sensor_data_in_2 = np.full(2048, 0)
async def handle_readData():
    await interrupt_0.wait()
    np.copyto(gradient_data_in, bram_gradient_reg_0)
    np.copyto(exc_current_data_in, bram_exc_current_reg_0)
    np.copyto(sensor_data_in_1, bram_in_reg_L_0)
    np.copyto(sensor_data_in_2, bram_in_reg_R_0)

asyncio.ensure_future(handle_readData())
```

```
#Create Fourier Components
def fullSpectrum(freq_spectrum):
    spectrum_mirror = np.full(1,0,dtype=complex)
    spectrum_mirror = np.append(spectrum_mirror, np.flip(np.conj(
        ↪ freq_spectrum[1:])))
    return np.concatenate((freq_spectrum, spectrum_mirror))

def generateSignal(frequency_spectrum_exc):
    return ifft(fullSpectrum(frequency_spectrum_exc)).real
```

```
#Generate signal with mixed frequencies
amplitude_spectrum_exc = np.full(1024,0)
```

```
for i in np.arange(1,60):
    amplitude_spectrum_exc[i] = 2048

phase_spectrum_exc = np.full(1024,0)
for i in np.arange(1,60):
    phase_spectrum_exc[i] = random.uniform(0, 2*math.pi)

frequency_spectrum_exc = amplitude_spectrum_exc * (np.cos(
  ↪ phase_spectrum_exc) + np.sin(phase_spectrum_exc) * 1j)

exc_signal = generateSignal(frequency_spectrum_exc)

# Generate Signal with correct Amplitude
amplitudeFactor = 0x27FF / np.amax(exc_signal)  # Max Amp. ~0
  ↪ x27FF
exc_signal = np.int32(np.multiply(amplitudeFactor, exc_signal))
print(exc_signal)
```

```
#A Function to set the amplitude of the excitation coil
def setAmplitude(factor, exc_signal):
    ## Scaling Amplitude of signal
    exc_signal = np.multiply(factor, exc_signal)
    np.copyto(bram_exc_reg, np.int32(exc_signal))
```

```
def readFlux1():
    return sensor_data_in_1

def readFlux2():
    return sensor_data_in_2

def readGradientFlux():
    return gradient_data_in

def readExcCurrent():
    return exc_current_data_in
```

```
def measurement(number):
    flux1 = np.full(2048,0,dtype=np.float32)
    flux2 = np.full(2048,0,dtype=np.float32)
    gradient_flux = np.full(2048,0,dtype=np.float32)
    exc_current = np.full(2048,0,dtype=np.float32)

    for i in range(number):
        asyncio.get_event_loop().run_until_complete(
          ↪ handle_readData())
        #print("interrupt")
        flux1_t = readFlux1()
        flux2_t = readFlux2()
```

```
        gradient_flux_t = readGradientFlux()
        exc_current_t = readExcCurrent()
        flux1 += flux1_t.astype(float)
        flux2 += flux2_t.astype(float)
        gradient_flux += gradient_flux_t.astype(float)
        exc_current += exc_current_t.astype(float)

    flux1 /= number
    flux2 /= number
    gradient_flux /= number
    exc_current /= number

    return flux1, flux2, gradient_flux, exc_current
```

```
def fourierTransform(flux1, flux2, gradient_flux, exc_current):
    n = np.arange(0,1024)
    frequencies = n*48000/2048
    return frequencies, fft(flux1), fft(flux2), fft(gradient_flux
      ↪ ), fft(exc_current)
```

```
#A function to calibrate the sensors with the current magnetic
  ↪ field induced by the excitation coil
def setReference(number):
    flux1_ref = np.full(2048,0,dtype=np.float32)
    flux2_ref = np.full(2048,0,dtype=np.float32)
    gradient_flux_ref = np.full(2048,0,dtype=np.float32)
    flux1_ref, flux2_ref, gradient_flux_ref, _ = measurement(
      ↪ number)

    return flux1_ref, flux2_ref, gradient_flux_ref
```

```
#Save the data to a JSON file
def complexToDict(z):
    return {"real":z.real, "imag":z.imag}
def saveAsJson(complex_array, file_path):
    serializable_array = [[complexToDict(z) for z in row] for row
      ↪  in complex_array]
    json.dump(serializable_array, codecs.open(file_path, "w",
      ↪ encoding="utf-8"),separators=(",", ":"), sort_keys=True
      ↪ , indent=4)
```

```
setAmplitude(4.5, exc_signal)
flux1_ref, flux2_ref, gradient_flux_ref = setReference(300)
print(flux1_ref)
print(flux2_ref)
```

```
#define arrays
c_ring_4_1 = np.full((10,2048),0,dtype=complex)
```

```
c_ring_6_05 = np.full((10,2048),0,dtype=complex)
c_ring_8_55 = np.full((10,2048),0,dtype=complex)
c_ring_13_75 = np.full((10,2048),0,dtype=complex)
sss = np.full((10,2048),0,dtype=complex)
ssm = np.full((10,2048),0,dtype=complex)
ssb = np.full((10,2048),0,dtype=complex)
distance = ["12.5","17.5","21.5","25.5","29.5","33.5","37.5","41.5","45.5","49.5"]
```

```
#Function to do one measurement after calibration
def getSpectrum(number, name, distance):
    lim = 45
    flux1, flux2, gradient_flux, exc_current = measurement(number)
    #flux1 -= flux1_ref
    #flux2 -= flux2_ref
    gradient_flux -= gradient_flux_ref
    frequencies, flux1_f, flux2_f, gradient_flux_f, exc_current_f = fourierTransform(flux1, flux2, gradient_flux, exc_current)
    ratio = flux2_f/flux1_f
    ratio -= fft(flux2_ref)/fft(flux1_ref)

    fig, axs = plt.subplots(2,2,figsize=(15,9))
    fig.suptitle(f"{name} at distance = {distance}cm")
    axs[0,0].plot(frequencies[1:lim],ratio.real[1:lim])
    axs[0,0].set_title("real part")

    axs[0,1].plot(frequencies[1:lim],ratio.imag[1:lim])
    axs[0,1].set_title("imaginary part")

    axs[1,0].plot(frequencies[1:lim],np.abs(ratio)[1:lim])
    axs[1,0].set_title("magnitude")

    axs[1,1].plot(frequencies[1:lim],np.angle(ratio)[1:lim])
    axs[1,1].set_title("phase")

    fig.subplots_adjust(wspace=0.5, hspace=0.5)  # Adjust these values as needed

    for ax in axs.flat:
        ax.grid(True)
    plt.show()
    return ratio
```

```
#Measurement for copper ring with radius 4.1cm
for i in range(10):
    for k in range(15):
```

```
        print(15-k)
        time.sleep(1)
    c_ring_4_1[i] = getSpectrum(300,"copper ring 4.1 cm radius",
      ↪ distance[i])
print("Measurement done")

#Measurement for copper ring with radius 6.05cm
for i in range(10):
    for k in range(15):
        print(15-k)
        time.sleep(1)
    c_ring_6_05[i] = getSpectrum(300,"copper ring 6.05 cm radius"
      ↪ ,distance[i])
print("Measurement done")

#Measurement for copper ring with radius 8.55cm
for i in range(10):
    for k in range(15):
        print(15-k)
        time.sleep(1)
    c_ring_8_55[i] = getSpectrum(300,"copper ring 8.55 cm radius"
      ↪ ,distance[i])
print("Measurement done")

#Measurement for copper ring with radius 13.75cm
for i in range(10):
    for k in range(15):
        print(15-k)
        time.sleep(1)
    c_ring_13_75[i] = getSpectrum(300,"copper ring 13.75 cm
      ↪ radius",distance[i])
print("Measurement done")

#Measurement for stainless steel small
for i in range(10):
    for k in range(15):
        print(15-k)
        time.sleep(1)
    sss[i] = getSpectrum(300,"stainless steel small",distance[i])
print("Measurement done")

#Measurement for stainless steel middle
for i in range(10):
    for k in range(15):
        print(15-k)
        time.sleep(1)
    ssm[i] = getSpectrum(300,"stainless steel middle",distance[i
      ↪ ])
```

```
print("Measurement done")

#Measurement for stainless steel big
for i in range(10):
    for k in range(15):
        print(15-k)
        time.sleep(1)
    ssb[i] = getSpectrum(300,"stainless steel big",distance[i])
print("Measurement done")

saveAsJson(c_ring_4_1, "./c_ring_4_1.json")
saveAsJson(c_ring_6_05, "./c_ring_6_05.json")
saveAsJson(c_ring_8_55, "./c_ring_8_55.json")
saveAsJson(c_ring_13_75, "./c_ring_13_75.json")
saveAsJson(sss, "./sss.json")
saveAsJson(ssm, "./ssm.json")
saveAsJson(ssb, "./ssb.json")
```

Appendix C Data Processing

```
from pdb import run
import matplotlib.pyplot as plt
import numpy as np
import json

from scipy.signal import savgol_filter
from scipy.ndimage import uniform_filter1d
import math
from scipy import stats

distance = ["12.5","17.5","21.5","25.5","29.5","33.5","37.5","
    ↪ 41.5","45.5","49.5"]

with open("c_ring_4_1.json", "r") as f:
    data = json.load(f)

c_ring_4_1 = np.array([[complex(item["real"], item["imag"])for
    ↪ item in row] for row in data])

with open("c_ring_6_05.json", "r") as f:
    data = json.load(f)

c_ring_6_05 = np.array([[complex(item["real"], item["imag"])for
    ↪ item in row] for row in data])

with open("c_ring_8_55.json", "r") as f:
    data = json.load(f)

```

```
c_ring_8_55 = np.array([[complex(item["real"], item["imag"])for item in row] for row in data])

with open("c_ring_13_75.json", "r") as f:
    data = json.load(f)

c_ring_13_75 = np.array([[complex(item["real"], item["imag"])for item in row] for row in data])

with open("ssb.json", "r") as f:
    data = json.load(f)

ssb = np.array([[complex(item["real"], item["imag"])for item in row] for row in data])

with open("ssm.json", "r") as f:
    data = json.load(f)

ssm = np.array([[complex(item["real"], item["imag"])for item in row] for row in data])

with open("sss.json", "r") as f:
    data = json.load(f)

sss = np.array([[complex(item["real"], item["imag"])for item in row] for row in data])

measurements = [sss, ssm, ssb, c_ring_4_1, c_ring_6_05, c_ring_8_55, c_ring_13_75]
label = ["sss","ssm","ssb","radius=4.1cm","radius=6.05cm","radius=8.55cm","radius=13.75cm"]


def running_average_filter(data, N, smoothing):
    # Define the filter kernel
    smoothed_data = data


    if (smoothing):
        #smoothed_data = uniform_filter1d(data,size=N) #running mean
        smoothed_data = savgol_filter(data, window_length=N, polyorder=3) #savgol filter

    return smoothed_data

lim = 45
for i in range(len(measurements)):
```

```
65      for k in range(len(measurements[i])):
66          measurements[i][k].real[1:lim] = running_average_filter(
               ↪ measurements[i][k].real[1:lim], N=11, smoothing=
               ↪ True)
67          measurements[i][k].imag[1:lim] = running_average_filter(
               ↪ measurements[i][k].imag[1:lim], N=11, smoothing=
               ↪ True)
68
69  def compareSizes(measurements, start, end):
70      lim = 45
71      fix, axs = plt.subplots(1,2,figsize=(15,9))
72      n = np.arange(0,1024)
73      frequencies = n*48000/2048
74      for i in np.arange(start, end):
75          axs[0].plot(frequencies[1:lim],measurements[i][0].real[1:
               ↪ lim],label=label[i])
76          axs[1].plot(frequencies[1:lim],measurements[i][0].imag[1:
               ↪ lim],label=label[i])
77          index = np.argmin(measurements[i][0].imag[0:lim])
78          print(frequencies[index])
79      axs[0].set_title("real part")
80      axs[1].set_title("imaginary part")
81      for ax in axs.flat:
82          ax.grid(True)
83          ax.set_xlabel("Frequency [Hz]")
84          ax.set_ylabel("Amplitude [arbitrary unit]")
85          ax.legend()
86      plt.show()
87
88  compareSizes(measurements, 0, 3) #compare spectra for stainless
       ↪ steel
89  compareSizes(measurements, 3, 7) #compare spectra for copper
       ↪ rings
90
91  def getMinima(measurement, number):
92      lim=45
93      frequency_minima = np.full(10,0)
94      value_at_freq_minima = np.full(10,0,dtype=np.float32)
95      for i in range(number):
96          imag_array = measurement[i].imag[0:lim]
97          index = np.argmin(imag_array)
98          frequency_minima[i] = index * 48000/2048
99          value_at_freq_minima[i] = imag_array[index]
100     return frequency_minima, value_at_freq_minima
101
102
103 #Function to scale all the 10 measurements of one specific metal
       ↪ on to the same amplitude
```

```
104 def scaleToSameAmplitude(measurement,label,number):
105     plt.figure()
106     lim = 38
107     n = np.arange(0,1024)
108     frequencies = n*48000/2048
109     index = np.argmin(measurement[0].imag[0:lim])
110     max_0 = measurement[0].imag[index]
111     for i in range(number):
112         imag_array = measurement[i].imag[0:lim]
113         index = np.argmin(imag_array)
114         scale = max_0/imag_array[index]
115         plt.plot(frequencies[1:lim], scale*imag_array[1:lim],
              ↪ label=distance[i]+"cm")
116 
117     plt.legend(loc='lower left', bbox_to_anchor=(0., 0.),
          ↪ borderaxespad=0.)
118     plt.grid(True)
119     plt.xlabel("Frequency [Hz]")
120     plt.ylabel("Amplitude [arbitrary unit]")
121     plt.title(label)
122 
123 
124 for i in np.arange(0,7):
125     scaleToSameAmplitude(measurements[i],label=label[i],number
          ↪ =10)
126 plt.show()
127 
128 
129 
130 def getAmplitudeAverage(measurement):
131     lim = 45
132     n = np.arange(0,1024)
133     frequencies = n*48000/2048
134     length = len(measurement[0].imag[1:lim])
135     #print(length)
136     value_imag = np.full(10,0,dtype=np.float32)
137     value_real = np.full(10,0,dtype=np.float32)
138     #print(measurement[0].imag[1:lim])
139     for k in range(10):
140         value_real_t = 0
141         value_imag_t = 0
142         for i in range(length):
143             value_imag_t += measurement[k].imag[i]
144             value_real_t += measurement[k].real[i]
145         value_real[k] = value_imag_t/length
146         value_imag[k] = value_real_t/length
147 
148     return np.abs(value_real), np.abs(value_imag)
```

```
149
150
151 value_real = np.full(10,0,dtype= np.float32)
152 value_imag = np.full(10,0,dtype= np.float32)
153 distance = [12.5,17.5,21.5,25.5,29.5,33.5,37.5,41.5,45.5,49.5]
154
155 #Plot value at minimum for different distances
156 fig, axs = plt.subplots(1,1,figsize=(15,9))
157 for i in range(len(measurements)):
158     frequency_minima, value_at_freq_minima = getMinima(
        ↪ measurements[i],10)
159     plt.plot([math.log10(x) for x in distance])
160     plt.title("value at minimum for different distances")
161     fit = stats.linregress([math.log10(x) for x in distance], [
        ↪ math.log10(x) for x in np.abs(value_at_freq_minima)])
162     print(label[i])
163     print(fit.slope)
164     print(fit.stderr)
165
166 plt.grid(True)
167 plt.legend()
168 plt.ylabel("log(A) [arbitrary unit]")
169 plt.xlabel("log(d) [cm]")
170
171 #Plot average amplitude for different distances
172 fig, axs = plt.subplots(1,2,figsize=(15,9))
173 for i in range(len(measurements)):
174     value_real, value_imag = getAmplitudeAverage(measurements[i])
175     fit_real = stats.linregress([math.log10(x) for x in distance
        ↪ ], [math.log10(x) for x in value_real])
176     fit_imag = stats.linregress([math.log10(x) for x in distance
        ↪ ], [math.log10(x) for x in value_imag])
177     print("real"+label[i])
178     print(fit_real.slope)
179     print(fit_real.stderr)
180     print("imag"+label[i])
181     print(fit_imag.slope)
182     print(fit_imag.stderr)
183     axs[0].plot([math.log10(x) for x in distance], [math.log10(x)
        ↪  for x in value_real], label=label[i])
184     axs[1].plot([math.log10(x) for x in distance], [math.log10(x)
        ↪  for x in value_imag], label=label[i])
185     axs[0].set_title("real part")
186     axs[1].set_title("imaginary part")
187
188 fig.subplots_adjust(wspace=0.5, hspace=0.5)  # Adjust these
    ↪ values as needed
189
```

```
for ax in axs.flat:
    ax.grid(True)
    ax.set_xlabel("log(d) [cm]")
    ax.set_ylabel("log(A) [arbitrary unit]")
    ax.legend()

plt.show()
```

Note of Thanks

I want to thank my supervisors Dr. Yves Acremann and Joel Rehmann for their patience and great support throughout the whole semester project. They were always available when I had questions. I learned many things regarding digital signal processing and embedded systems and working with FPGAs definitely broadened my horizon of what is technologically possible.
I also got to experience what real experimental physicists are doing every day from their scientific research all the way to their humor.
I would also like to thank Robert Früh, with whom working with was a great source of inspiration.
I hope this project will eventually find use in humanitarian demining and possibly help people or even save lives.