Peter Luidolt, BSc

# Realization of a Set-Up for Hall Effect Measurements

**MASTER'S THESIS**

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Technical Physics

submitted to

**Graz University of Technology**

Supervisor

Ao.Univ.-Prof. Dipl-Ing. Dr.techn. Roland Resel

Institute of Solid State Physics

Graz, July 2017

# AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

# Acknowledgements

First, I want to thank my supervisor Roland Resel for giving me the opportunity to work on this project and the possibility to be part of his team. He always supported me in every aspect and made it possible to finish my work in the narrow time frame I had.

It was a great experience to work in this research group and I want to especially thank Stefan Pachmajer, Andrew Jones, Fabian Muralter and Martin Tazreiter for many discussions and the continuous help with upcoming riddles, problems and challenges.

I also want to thank Peter Hadley for advice on many topics and always having an open door for me as well as Martin Kornschober for helping me during the whole process of the laboratory setup and providing me with insight into the mechanical construction process. I highly appreciate the help from Alberto Perrotta for performing spectroscopic ellipsometry measurements and also help from Harald Kerschbaumer, Birgit Kunert, Elisabeth Stern and Robert Schennach is gratefully appreciated.

The opportunity to collaborate with the Russian Academy of Sciences, Ioffe Physical Technical Institute was a great experience and I want to specially thank Alexander Burkov and P.P. Konstantinov for performing Hall effect and resistivity measurements on prepared samples and providing me with insight about their Hall effect measurement setup.

I want to thank my parents Herbert and Inge and my brothers Christian and Markus for their continuous support, discussions and patience for so many years.

Finally, I want to thank Katrin - the love of my life - for dreaming, laughing and sharing the most valuable and exiting things in life with me.

Thank you Ria, for giving me the final push!

# Abstract

Hall effect measurements are a well established and widely used method in the semiconductor industry to gain information about the electrical properties of materials such as electrical resistivity, charge carrier type, charge carrier density and charge carrier mobility. In this work, the process of setting up a new Hall effect measurement laboratory at the Institute of Solid State Physics at the Graz University of Technology is shown. The setup provides the possibility to perform simultaneous measurement of electrical resistivity and Hall coefficient and uses a combined AC / DC method to enhance accuracy. The setup can provide a constant magnetic field of up to 1 T with a homogeneity better than 99.9 % and is capable of performing measurements on materials with an electrical resistance between 0.1 Ω and 10 MΩ. Besides performance verification of existing devices, also a new cryostat was acquired and characterized in order to perform temperature dependent measurements in the range of 8 K to 800 K. For all measurement devices, a Python library was developed to provide easy and future-proof access to device functionalities. To verify the performance of the new laboratory, custom Indium Tin Oxide (ITO) Hall geometries have been laser structured with an accuracy of ±2 μm. Temperature dependent measurements of the ITO between 8 K and 300 K reveal an increase of electrical resistivity with rising temperature. For room temperature measurements, the produced ITO standard samples show a resistivity of $\rho = 184\,\mu\Omega\,\mathrm{cm}$, a charge carrier density of $n = 9.33\;10^{20}\,\mathrm{cm}^{-3}$ and a charge carrier mobility of $\mu = 36.35\,\mathrm{cm}^2/(\mathrm{V\,s})$. Those values are in good agreement with results from measurements performed at the Russian Academy of Sciences, Ioffe Physical Technical Institute.

# Kurzfassung

Hall Effekt Messungen sind in der Halbleiterindustrie eine etablierte und weit verbreitete Methode zu Bestimmung von elektrischen Materialparametern wie dem elektrische Widerstand, dem Ladungsträgertyp, der Ladungsträgerdichte und der Ladungsträgerbeweglichkeit. In dieser Arbeit wird der Aufbau eines neuen Hall Effekt Messlabors am Institut für Festkörperphysik der Technischen Universität Graz gezeigt. Das aufgebaute Setup bietet die Möglichkeit gleichzeitig den elektrischen Widerstand sowie den Hall Koeffizienten zu messen und benutzt dabei eine kombinierte AC / DC Methode um die Messgenauigkeit zu erhöhen. Messungen an Proben mit einem elektrischen Widerstand zwischen 0.1 Ω und 10 MΩ können bei Magnetfeldern bis zu 1 T durchgeführt werden. Neben der Charakterisierung von bestehenden Geräten wurde auch ein neues Kryostat angeschafft um temperaturabhängige Messungen in einem Bereich von 8 K bis 800 K durchführen zu können. Für alle Messgeräte wurde eine Python Bibliothek programmiert die eine einfache und zukunftssichere Ansteuerung der Messgeräte ermöglicht. Zur Verifizierung des neuen Labors wurden spezielle Indium Zinn Oxid (ITO) Hall Strukturen mittels Laser-Strukturierung hergestellt. Der ausgewählte Herstellungsprozess garantiert Strukturgenauigkeiten von ±2 µm. Temperatur abhängige Messungen zwischen 8 K und 300 K zeigen einen steigenden spezifischen Widerstand mit steigender Temperatur. Für Raumtemperatur zeigen die ITO Standard Proben einen spezifischen Widerstand von $\rho = 184\,\mu\Omega\,\mathrm{cm}$, eine Ladungsträgerdichte von $n = 9.33\ 10^{20}\,\mathrm{cm}^{-3}$ und eine Ladungsträgerbeweglichkeit von $\mu = 36.35\,\mathrm{cm}^2/(\mathrm{V\,s})$. Die ermittelten Werte zeigen eine gute Übereinstimmung mit Messergebnissen des Russian Academy of Sciences, Ioffe Physical Technical Institute.

# Contents


**Acknowledgements** v

**Abstract** vii

**Abstract** ix

**1. Introduction** 1

**2. Fundamentals** 3
- 2.1. The Drude-Model . . . 3
- 2.2. The Hall Effect . . . 4
- 2.3. Hall geometries . . . 7
  - 2.3.1. Van der Pauw geometries . . . 7
  - 2.3.2. Hall bar geometries . . . 8
- 2.4. Indium Tin Oxide (ITO) . . . 10

**3. Measurement techniques** 13
- 3.1. Van der Pauw . . . 13
- 3.2. Hall bar type . . . 19
- 3.3. Low level current measurement . . . 20
- 3.4. Low level voltage measurement . . . 23

**4. Experimental Section** 27
- 4.1. Sample Preparation . . . 27
  - 4.1.1. Geometry Considerations . . . 28
  - 4.1.2. Sample Investigation . . . 30
- 4.2. Laboratory Setup . . . 31
  - 4.2.1. Magnet Characterization . . . 31
  - 4.2.2. Closed Cycle Cryostat . . . 34
  - 4.2.3. Programming of measurement devices . . . 40
- 4.3. Measurement setup . . . 51
  - 4.3.1. Setup at the Graz University of Technology . . . 51
  - 4.3.2. Setup at the Ioffe Institute . . . 53

**5. Results** 55

6. Conclusions 61

A. Appendix 65
A.1. Device specifications . . . . . . . . . . . . . . . . . . . . . 65
A.1.1. Bruker Electromagnet . . . . . . . . . . . . . . . . . 65
A.1.2. Heinzinger Power Supply . . . . . . . . . . . . . . . 66
A.1.3. Advanced Research Systems Cryostat . . . . . . . 69
A.1.4. Keithley SourceMeter 2600 . . . . . . . . . . . . . . 91
A.1.5. Agilent Switch Mainframe . . . . . . . . . . . . . . 91
A.1.6. Magnet-Physic magnetometer . . . . . . . . . . . . 95
A.1.7. Lock-In Amplifiers . . . . . . . . . . . . . . . . . . . 97
A.1.8. Function generator . . . . . . . . . . . . . . . . . . . 104
A.2. Source codes . . . . . . . . . . . . . . . . . . . . . . . . . 108
A.2.1. Calculation of the van der Pauw geometry correction factor . . . . . . . . . . . . . . . . . . . . . 108
A.2.2. Van der Pauw measurement using four Source Measure Units . . . . . . . . . . . . . . . . . . . . . 108
A.2.3. 2D-Stage Stepper Control code . . . . . . . . . . . 116
A.2.4. Agilent 3499A switch mainframe library . . . . . 121
A.2.5. Stanford Research Systems SR830 lock-in Python library . . . . . . . . . . . . . . . . . . . . . . . . . 122
A.2.6. Princeton Applied Research Model 5210 lock-in Python library . . . . . . . . . . . . . . . . . . . . . . 127
A.2.7. Heinzinger PTN40-125 Power Supply . . . . . . . 131
A.2.8. MagnetPhysik FH54 Magnetometer . . . . . . . . 133
A.2.9. Philips PM5193 function generator . . . . . . . . . 138
A.2.10. Keithley 199 Multimeter . . . . . . . . . . . . . . . . 141
A.2.11. Keithley SourceMeter 2600 series . . . . . . . . . . 143
A.2.12. Hall effect and resistivity measurement . . . . . . 156

Bibliography 165

# List of Figures


2.1. Direction of the most important physical quantities for measurement of Hall effect . . . . . . . . . . . . . . . . . . . 4
2.2. Resistivity measurement on Hall bar type sample . . . . 6
2.3. Van der Pauw measurements on arbitrary shape . . . . . 8
2.4. Hall measurement - van der Pauw shapes . . . . . . . . . 8
2.5. Hall measurement - bridge-type . . . . . . . . . . . . . . 9
2.6. Optical transmission of ITO . . . . . . . . . . . . . . . . . 11

3.1. Flat lamella of arbitrary shape . . . . . . . . . . . . . . . 13
3.2. Van der Pauw geometrical correction factor . . . . . . . . 15
3.3. Multiple van der Pauw resistivity measurements . . . . 15
3.4. Multiple van der Pauw Hall measurements . . . . . . . . 16
3.5. Typical van der Pauw measurement setup . . . . . . . . 17
3.6. Van der Pauw measurement configuration with four Source Measure Units . . . . . . . . . . . . . . . . . . . . . 18
3.7. Hall bar geometry . . . . . . . . . . . . . . . . . . . . . . . 18
3.8. Step function test . . . . . . . . . . . . . . . . . . . . . . . 21
3.9. Guarding the Leakage Resistance of a Cable . . . . . . . 22
3.10. Experimental setup of a lock-in amplifier . . . . . . . . . 24

4.1. Laser structured Hall geometries . . . . . . . . . . . . . . 28
4.2. Hall bar custom shape . . . . . . . . . . . . . . . . . . . . 29
4.3. Dimensions of selected ITO Hall geometries . . . . . . . 30
4.4. Spectroscopic ellipsometry measurement . . . . . . . . . 31
4.5. Bruker Magnet type B-E15 B8 . . . . . . . . . . . . . . . . 32
4.6. 2D-stage for Hall sensor movement . . . . . . . . . . . . 33
4.7. 2D magnetic field homogeneity measurement . . . . . . 33
4.8. 1D magnetic field homogeneity measurement . . . . . . 34
4.9. Cold head of the ARS cryostat . . . . . . . . . . . . . . . 35
4.10. Cryostat cool down characteristic . . . . . . . . . . . . . . 36
4.11. Cryostat self heat up characteristic . . . . . . . . . . . . . 36
4.12. SolidWorks rendering of the closed cycle cryostat support frame. . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.13. Support frame for the closed cycle cryostat. . . . . . . . . 38
4.14. LakeShore 336 temperature controller . . . . . . . . . . . 39
4.15. Structure of device control using Python . . . . . . . . . 40

4.16. Switch matrix configuration . . . . . . . . . . . . . . . . . . 42
4.17. Four quadrants of operation . . . . . . . . . . . . . . . . . . 46
4.18. Correct use of setpoint feature . . . . . . . . . . . . . . . . 50
4.19. Measurements setup at the TUGraz . . . . . . . . . . . . 52
4.20. Flow diagram of Hall and resistivity measurements . . . 52
4.21. Hall bar specimen on sample holder . . . . . . . . . . . . 53
4.22. Measurements setup at the Ioffe Institute . . . . . . . . . 54

5.1. ITO measurement (70 K to 500 K) performed at the Ioffe Institute . . . . . . . . . . . . . . . . . . . . . . . . . . 55
5.2. Resistivity comparison Ioffe and TUGraz . . . . . . . . . 56
5.3. ITO measurement (300 K to 620 K) performed at the Ioffe Institute . . . . . . . . . . . . . . . . . . . . . . . . . 57
5.4. ITO contact hairline fracture . . . . . . . . . . . . . . . . . 58
5.5. Hall coefficient comparison Ioffe and TUGraz . . . . . . 58
5.6. Resistivity, carrier concentration and Hall mobility of ITO 59

A.1. Dimensions of the Bruker B-E15 electromagnet . . . . . . 65
A.2. Principle schematic of the Heinzinger PTN 125-40 [29] . 66
A.3. Components of the ARS cold-head . . . . . . . . . . . . . 75
A.4. Typical setup of the ARS cryostat components . . . . . . 75
A.5. Source and measure limits of the Keithley SMU2600 series 91
A.6. Schematic of the 44473A matrix switch module . . . . . 94
A.7. FH-54 magnetometer technical specifications . . . . . . . 95

# 1. Introduction

Silicon is nowadays the most widely used semiconductor material in electronic devices. Silicon is cheap, robust and easy to process but has some limitations where other materials need to be used [1]. With the rapid development in electronics, the need for new materials with specially tuned properties is immanent. Promising candidates such as doped zinc oxide (ZnO) are investigated by different groups [2], [3], [4].

Upon fabrication of new materials, it is vital to know electrical material properties such as resistivity, charge carrier type, charge carrier concentration and charge carrier mobility. Despite Hall effect measurements have been known since 1879 [5], this method of investigation is still one of the industries standard to test materials for their electrical properties. As testing methods and instruments have vastly improved it is possible to investigate the whole span of materials from high conductive, highly doped materials up to nearly pure, low conductive semiconductors [6]. Although experimental techniques have improved, Hall effect measurement on semiconductors can still be a challenge and requires careful interpretation of measurement results [7], [8].

The aim of this work was to establish the possibility to perform Hall effect measurements at the Institute of Solid State Physics at the University of Technology, Graz. This work will give an insight about the fundamentals of Hall effect measurements, the measurement methods and devices needed to perform those measurements, provide information how the new laboratory was build and how standard samples were prepared to get first performance results.

# 2. Fundamentals

## 2.1. The Drude-Model

The Drude-Model describes the classical charge transport inside a material caused by an external electric field [9]. In the Drude-Model a conductor is seen as an ion-crystal with free moving electrons that form an electron-gas. If an external electric field $\vec{E}$ is applied, the electrons inside the conductor experience the force:

$$\vec{F} = q\,\vec{E} \tag{2.1}$$

This force causes an acceleration of the electrons until an equilibrium is reached with a mean electron velocity resulting in an electrical current proportional to the strength of the electric field. This equilibrium was explained by Drude due to the assumption, that moving electrons collide with ions and will be decelerated. With the introduction of the mean time $\tau$ between two collisions, the equation of motion can be written as (with the electron mass $m$, the electron-velocity $v$ and the electron-drift-velocity $v_D$):

$$m\,\dot{v} + \frac{m}{\tau}\,v_D = -e\,E \tag{2.2}$$

In the stationary state ($\dot{v} = 0$) equation 2.2 can be rearranged and brought into relation with the current density $j$:

$$j = -e\,n\,v_D = \frac{e^2\,\tau\,n}{m}\,E \tag{2.3}$$

Equation 2.3 shows, that the current density $j$ is linear dependent on the charge carrier density $n$. With known current density, the conductivity $\sigma$ can be calculated:

$$\sigma = \frac{j}{E} = \frac{e^2\,\tau\,n}{m} \tag{2.4}$$

The Drude model can be also applied to positive charge carriers (holes) in the same way. In the Drude model the interactions between the charge carriers (electrons or holes) themselves are not taken into account which was later improved by Sommerfeld who described charge movement as a Fermi-gas [10] instead of a classical ideal gas as it is described in the Drude model. Nevertheless the Drude theory provides a good explanation for the Hall effect that will be used in the following chapter.

## 2.2. The Hall Effect

When current flows through a conductor and a stationary magnetic field is present in perpendicular direction, the charge carriers inside the conductor experience a force in transverse direction. This force leads to a charge carrier imbalance inside the conductor resulting in an electric field that can be measured as Hall voltage.

Edwin Hall showed this effect in 1879 on a gold sample [5]. This was the first proof that charge carries inside metals are moving electrons and not protons.

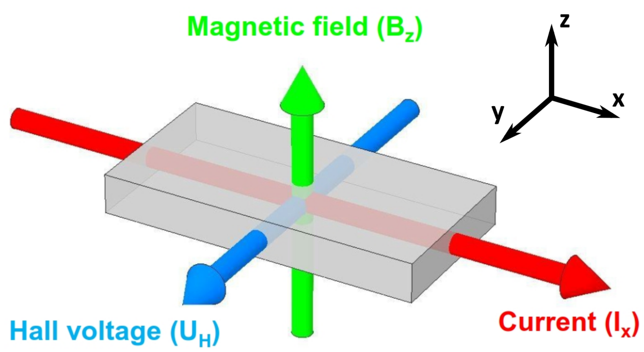



Figure 2.1.: Direction of the most important physical quantities for measurement of Hall effect

The Hall effect can be described by the Lorentz force acting on charge carriers inside the material.

$$\vec{F} = q(\vec{E} + (\vec{v} \times \vec{B})) \tag{2.5}$$

For the following calculations the coordinate system is defined so, that the current is flowing along the x-axis $I_x$ (the drift velocity is therefore

Table 2.1.: Definition of quantities in order of their occurrence

| **quantities** | **symbol** | **unit** |
|---|---|---|
| current density | $\vec{j}$ | $\mathrm{A\,m^{-2}}$ |
| current | $I$ | $\mathrm{A}$ |
| area | $A$ | $\mathrm{m^2}$ |
| charge carrier density | $n$ | $\mathrm{A\,s\,m^{-3}}$ |
| charge | $q$ | $\mathrm{A\,s}$ |
| drift velocity | $\vec{v}$ | $\mathrm{m\,s^{-1}}$ |
| force | $\vec{F}$ | $\mathrm{kg\,m\,s^{-2}}$ |
| electric field | $\vec{E}$ | $\mathrm{V\,m^{-1}}$ |
| magnetic field | $\vec{B}$ | $\mathrm{N\,m^{-1}\,A^{-1}}$ |
| width | $a$ | $\mathrm{m}$ |
| thickness | $t$ | $\mathrm{m}$ |
| Hall voltage | $U_H$ | $\mathrm{V}$ |
| Hall coefficient | $R_H$ | $\mathrm{m^3\,C^{-1}}$ |
| mobility | $\mu$ | $\mathrm{m^2\,V^{-1}\,s^{-1}}$ |
| elementary charge | $e$ | $\mathrm{A\,s}$ |
| distance | $d$ | $\mathrm{m}$ |
| electrical resistivity | $\rho$ | $\Omega\,\mathrm{m}$ |

also along the x-axis $\vec{v} = (v_x, 0, 0))$ and the magnetic field is applied along the z-axis $\vec{B} = (0, 0, B_z)$ leading to an electric field in y-direction $\vec{E} = (0, E_y, 0)$ as shown in figure 2.1.

In steady state, the generated electric field compensates for the force caused by the magnetic field, resulting in $F = 0$. The Lorentz force can be rewritten as:

$$E_y - v_x B_z = 0 \tag{2.6}$$

Due to the miss-balance of charge carriers, the specimen can be treated as capacitor and the electric field can be written as:

$$E_y = U_H a \tag{2.7}$$

The current density through a conductor can be described as:

$$\vec{j} = \frac{I}{a \cdot t} = nq\vec{v} \tag{2.8}$$

Equation 2.6 combined with 2.7 and 2.8 states the relation between the measured Hall voltage ($U_H$) and the Hall constant $R_H$. The calculation of the carrier density ($n$) is only valid for materials with one major carrier type.

$$U_H = R_H \frac{I_x B_z}{t} \tag{2.9}$$

$$R_H = \frac{1}{nq} \tag{2.10}$$

A negative Hall voltage ($U_H$) indicates a n-type material (electrons as charge carriers) whereas a positive Hall voltage indicates p-type material (holes as charge carriers).

To calculate the mobility of the charge carriers, additionally the resistivity of the material has to be determined. The resistivity of the specimen can be calculated by measurement of the voltage drop ($U_R$) across a known distance $d$ as shown in figure 2.2. To avoid errors due to the contact resistance, the measurement is performed by use of distinct contacts resulting in a four contact resistivity measurement.

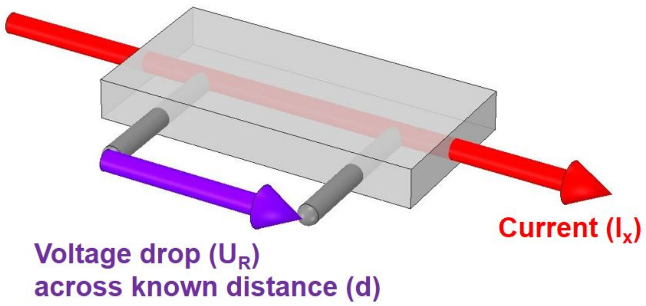



Figure 2.2.: Resistivity measurement on a Hall bar sample type by measurement of the voltage drop across contacts of known geometry.

$$\rho = \frac{U_R}{I_x} \frac{A}{d} \tag{2.11}$$

When a magnetic field is applied during resistivity measurement, the magnetoresistance $\rho(B_Z)$ of the specimen can be obtained.

For materials with dominant carrier type and known resistivity ($\rho$) and Hall coefficient ($R_H$), the carrier density ($n$) and carrier mobility ($\mu_H$) can be calculated:

$$n = -\frac{1}{R_H \cdot q} \tag{2.12}$$

$$\mu_H = \frac{|R_H|}{\rho} \tag{2.13}$$

In semiconductors, where the material often has both holes and electrons as charge carriers, the Hall constant needs to include the different carrier concentrations and carrier mobilities. With index $p$ indicating the quantity is related to positive charge carriers and index $e$ indicating relation to negative charge carriers.

$$R_H = \frac{n_p \mu_p^2 - n_e \mu_e^2}{e(n_p \mu_p + n_e \mu_e)^2} \tag{2.14}$$

## 2.3. Hall geometries

### 2.3.1. Van der Pauw geometries

Van der Pauw discovered that it is possible to measure the sheet resistance and the Hall coefficient of a thin, uniform sample of arbitrary shape [11].

This chapter will focus on special van der Pauw geometries that are frequently used when performing Hall measurements and explain why it is beneficial to use those geometries. Details to the van der Pauw measurement method will be discussed in chapter 3.1.

To perform accurate van der Pauw measurements, the specimens need to fulfill the following requirements [11]:

- The sample has to be a thin film with $t < 0.1$ cm.
- The sample needs to be homogeneous (no holes).
- The contacts need to be placed on the edges of the sample.
- The contacts must be point-like.

In practice point-like contacts can not be achieved and measurements will produce erroneous results. To overcome this limitation, special van der Pauw geometries can be used to minimize the influence of finite sized contacts. In figure 2.4 the most often used structures are listed.

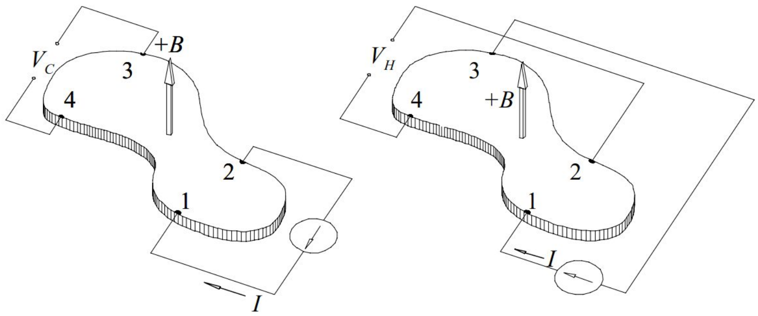



Figure 2.3.: Representation of the two necessary Van der Pauw measurements on a arbitrary geometry to calculate sheet resistance [12].

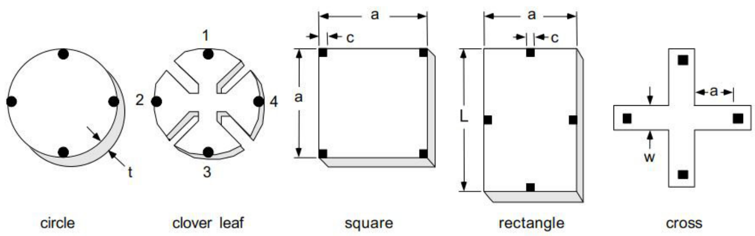



Figure 2.4.: Commonly used van der Pauw sample geometries to perform Hall effect and resistivity measurements. Modified from [12].

For Hall effect measurements, the clover leaf or the cross structure as shown in figure 2.4 reduce the error caused by finite contact size the most. The American Society for Testing and Materials released the standard *ASTM F76-08: Test Methods for Measuring Resistivity and Hall Coefficient and Determining Hall Mobility in Single-Crystal Semiconductors* [13]. This standard recommends, that the sample thickness is uniform to $\pm 1\%$ and should not be bigger than 0.1 cm. The recommended ratio of length $L$ to thickness $t$ should be $b \geq 15 \cdot t$. Contacts should be placed on the edges of the sample and the contact size must be smaller than $0.05 \cdot L$.

A limitation of the van der Pauw geometry is, that it is not possible to perform accurate magnetoresistance measurements and that it will produce erroneous results when used on anisotropic materials.

### 2.3.2. Hall bar geometries

The second class of sample geometries, that are used for Hall effect measurements, are called parallelepiped, bridge-type or Hall bar geometries. In contrast to the van der Pauw geometries these samples need to fulfill higher accuracy in shape.

Table 2.2.: Geometry definitions of bridge-type specimens. Eight-contact geometry as shown in figure 2.5(a) and 2.5(c) [13].

| |
|---|
| $L \geq 4 \cdot w$ |
| $w \geq 3 \cdot a$ |
| $b1, b2 \geq w$ |
| $t \leq 0.1\,\text{cm}$ |
| $c \geq 0.1\,\text{cm}$ |
| $1.0\,\text{cm} \leq L \leq 1.5\,\text{cm}$ |
| $b_1 = b_1' \pm 0.005\,\text{cm}$ |
| $b_2 = b_2' \pm 0.005\,\text{cm}$ |
| $d_1 = d_1' \pm 0.005\,\text{cm}$ |
| $d_2 = d_2' \pm 0.005\,\text{cm}$ |
| $b_1 + d_1 = (1/2) \cdot L + 0.005\,\text{cm}$ |
| $b_1' = d_1' = (1/2) \cdot L \pm 0.005\,\text{cm}$ |
| $b_1 \approx b_2, d_1 \approx d_2$ |

The measurement techniques used for parallelepiped samples are described in chapter 3.2.

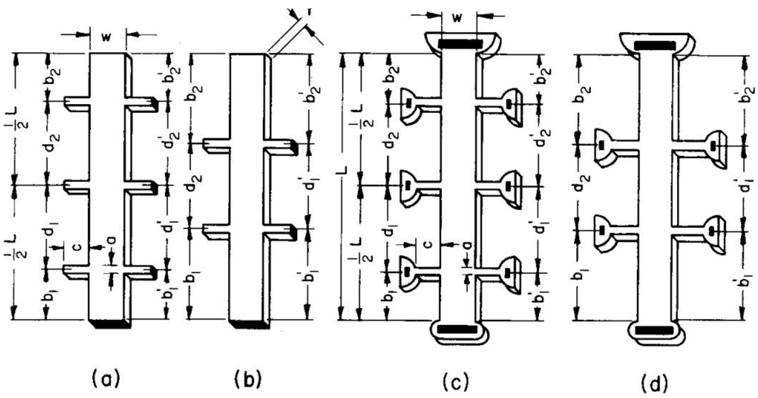



Figure 2.5.: Typical bridge-type specimen geometries to perform Hall effect and resistivity measurements [13].

Typical bridge-type samples are shown in figure 2.5. In table 2.2 and table 2.3 the geometry specifications for specimen preparation are listed as described in the ASTM F76-08 standard [13].

The major benefit of these samples is, that finite contact size has less effect on the measurement result. Also these type of geometries are better suited for resistivity measurements on high conductive samples. The larger distance between the measurement contacts (compared to van der Pauw geometries) will cause a larger voltage to be present

Table 2.3.: Geometry definitions of bridge-type specimens. Six-contact geometry as shown in figure 2.5(b) and 2.5(d) [13].

| |
|---|
| $L \geq 5 \cdot w$ |
| $w \geq 3 \cdot a$ |
| $b1, b2 \geq 2 \cdot w$ |
| $t \leq 0.1\,\text{cm}$ |
| $c \geq 0.1\,\text{cm}$ |
| $1.0\,\text{cm} \leq L \leq 1.5\,\text{cm}$ |
| $b_1 = b_1' \pm 0.005\,\text{cm}$ |
| $b_2 = b_2' \pm 0.005\,\text{cm}$ |
| $d_2 = d_1' \pm 0.005\,\text{cm}$ |
| $b_1 \approx b_2$ |

between the voltage measurement contacts. This is especially important if magnetoresistance measurements should be performed because of the small change in resistance that needs to be detected reliably.

A disadvantage - apart from the more complicated sample geometry - is, that more electrical connections are needed to be able to perform Hall and resistivity measurements.

To make contacting easier it is recommended to extend the side arms of the sample as shown in figure 2.5(c) and 2.5(d). These contacts on the side should be less than 0.02 cm in width. This however makes the sample more fragile.

## 2.4. Indium Tin Oxide (ITO)

Indium Tin Oxide (ITO) is a heavily doped n-type material that usually consists of 90 % Indium(III)-oxide ($In_2O_3$) and 10 % Tin(IV)-oxide ($SnO_2$) [14]. It has a wide bandgap of around 4 eV that makes it mostly transparent in the visible range [15].

ITO is widely used in nowadays industry and electronic devices because of being electrical conductive and being transparent in the visible range as shown in figure 2.6. One of the main applications of ITO is the use as electrode in liquid crystal displays.

ITO is usually coated onto glass substrate. There are several deposition techniques such as chemical vapor deposition [16], magnetron sputtering [17], evaporation [18] among others.

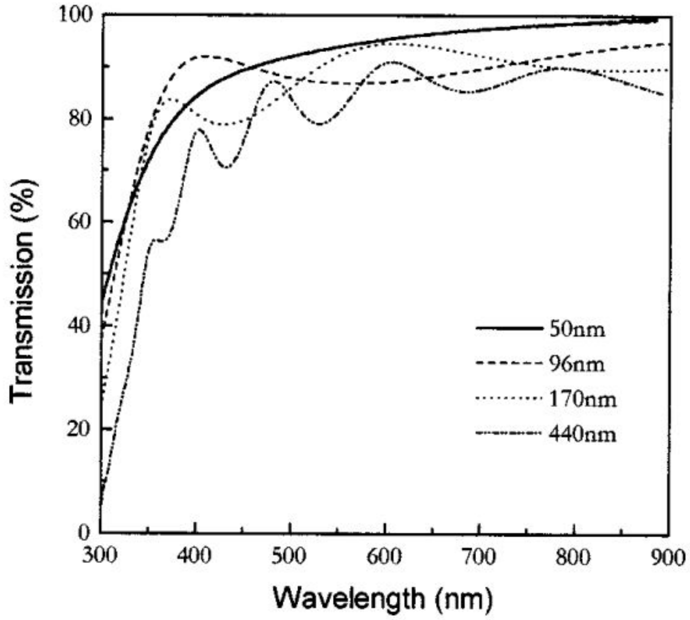



Figure 2.6.: Effect of film thickness on the optical transmission for the films grown at 300 °C in 10 mTorr of oxygen [15].

The use in modern electronics most often requires structuring. For mass production photo-lithography and etching is the common technique. For smaller quantities as well as higher accuracies, laser structuring can be used.

For verification and performance tests of the new build Hall measurement laboratory, ITO was chosen as reference material because of its well known characteristics as well as the possibility to have different geometries structured at high accuracy. ITO is stable up to approximately 500 K and has a high conductivity. Details to the produced standard samples will be discussed in chapter 4.1.1.

# 3. Measurement techniques

## 3.1. Van der Pauw

Van der Pauw geometries require multiple measurements in order to be able to calculate the sheet resistance. This chapter will concentrate on the measurement techniques needed to perform resistivity and Hall effect measurements on those type of samples.

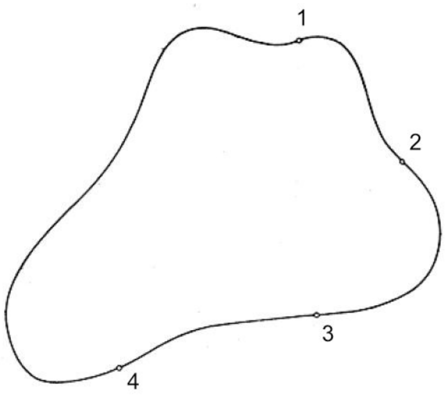



Figure 3.1.: Flat lamella of arbitrary shape, with four contacts 1, 2, 3 and 4 on the periphery modified from [11].

In the following notation the indices describe the used contacts. $I_{AB}$ specifies a current-flow between contact A and B. Likewise $U_{CD}$ refers to the potential difference $U_C - U_D$ measured between contact C and D.

A van der Pauw resistivity measurement consists of at least two measurements. First a resistance $R_{43,21}$ measurement is taken by providing a current $I_{43}$ and measuring the voltage $U_{21}$.

$$R_{43,21} = \frac{U_{21}}{I_{43}} \tag{3.1}$$

Analogously a second measurement is taken:

$$R_{32,14} = \frac{U_{14}}{I_{32}} \tag{3.2}$$

As van der Pauw showed, the electrical resistivity $\rho$ can be calculated by the following relationship between the two resistance measurements taken if the sheet thickness $d$ is known [11].

$$\exp\left(-\frac{\pi d}{\rho} \cdot R_{43,21}\right) + \exp\left(-\frac{\pi d}{\rho} \cdot R_{32,14}\right) = 1 \tag{3.3}$$

For an arbitrary shape, an expression for $\rho$ can not be gained analytically. However it is possible to rearrange to the following form with introduction of the correction factor $f$:

$$\rho = \frac{\pi d}{\ln 2} \cdot \frac{R_{43,21} + R_{32,14}}{2} \cdot f \tag{3.4}$$

The factor $f$ can be expressed as a function of the ratio of the two measured resistances as shown in equation 3.6 and can be calculated numerically. The result is shown in graph 3.2. This numerical calculation was done using `fsolve` from the Python `scipy.optimize` package. The source to the calculation can be found in appendix A.2.1.

$$Q := \frac{R_{43,21}}{R_{32,14}} \tag{3.5}$$

$$\frac{Q-1}{Q+1} = \frac{f}{\ln(2)} \cdot \cosh\left(\frac{1}{2} \cdot \exp\left(\frac{\ln(2)}{f}\right)\right) \tag{3.6}$$

As shown, only two measurements are sufficient to obtain the resistivity $\rho$. To get a higher accuracy, it is beneficial to take more measurements as illustrated in figure 3.3. To eliminate offset errors from measurement instruments, it is recommended to perform all the measurements shown in figure 3.3 with reversed polarity as well. This leads to a total of eight measurements to gain accurate sheet resistivity.

With the van der Pauw geometry also Hall measurements can be performed. Therefore a current is driven through opposite contacts (for example contacts 4 and 2) and a resistance measurement $R_{42,31}$ is performed by measuring the voltage across the two remaining contacts (3 and 1). After this initial measurement a homogeneous magnetic

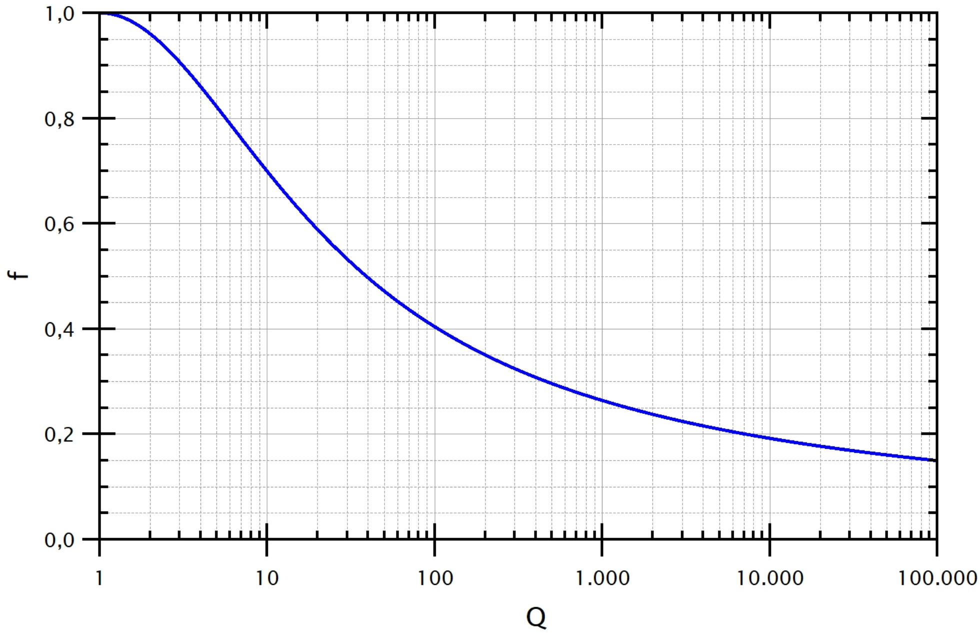



Figure 3.2.: Numerically calculated van der Pauw geometrical correction factor needed for calculation of the resistivity ρ by use of equation 3.4.

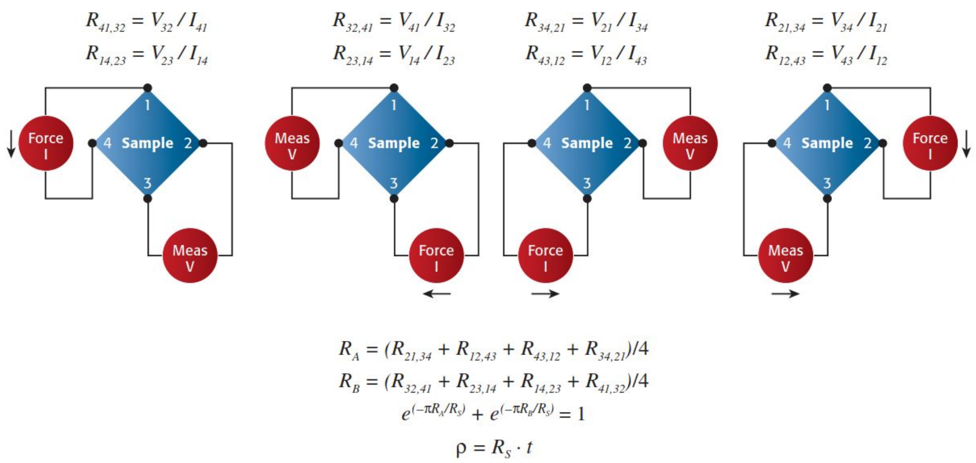



Figure 3.3.: Computing average resistivity ρ with multiple van der Pauw measurements [6].

field B perpendicular to the surface is applied and the measurement is done again. The resulting change in resistance is $\Delta R_{42,31}$ and the Hall coefficient can be written as:

$$R_H = \frac{d}{B} \cdot \Delta R_{42,31} \tag{3.7}$$

Depending on the geometry, the change of resistance ($\Delta R_{42,31}$) can be small compared to the absolute values of $R_{42,31}$. This means that the voltages that need to be measured ($U_{31}$ with and without magnetic field) require the measurement equipment to be set to a high enough range so that the input does not saturate. If the voltage difference is small, this may be a problem because the dynamic measurement range of the equipment might not be sufficient to reliably measure the small difference in the signals.

To minimize this effect, it is beneficial to use van der Pauw geometries that have a high symmetry. For semiconductor characterization often the clover leaf or the Greek cross geometry is used as shown in figure 2.4.

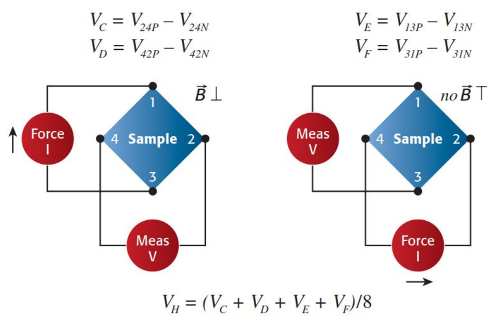



Figure 3.4.: Computation of the Hall voltage with both positive and negative polarity current and with the magnetic field up and down [6].

To enhance accuracy, this measurement should be repeated with permuted contacts ($\Delta R_{31,42}$) as well as with reversed polarity and average over the measured quantities. A whole of four measurements are taken as shown in figure 3.4.

To further improve confidence in the measurement results, it is recommended to repeat these four Hall measurements with 180° rotated magnetic field.

If temperature dependent van der Pauw measurements are done, the temperature needs to be kept stable during the series of measurements. It is recommended by the ASTM F76-08 standard [13] to perform temperature measurements at least before and after the resistivity and Hall measurements. If the temperature change exceeds ±1 °C the resistivity and Hall measurement should be repeated.

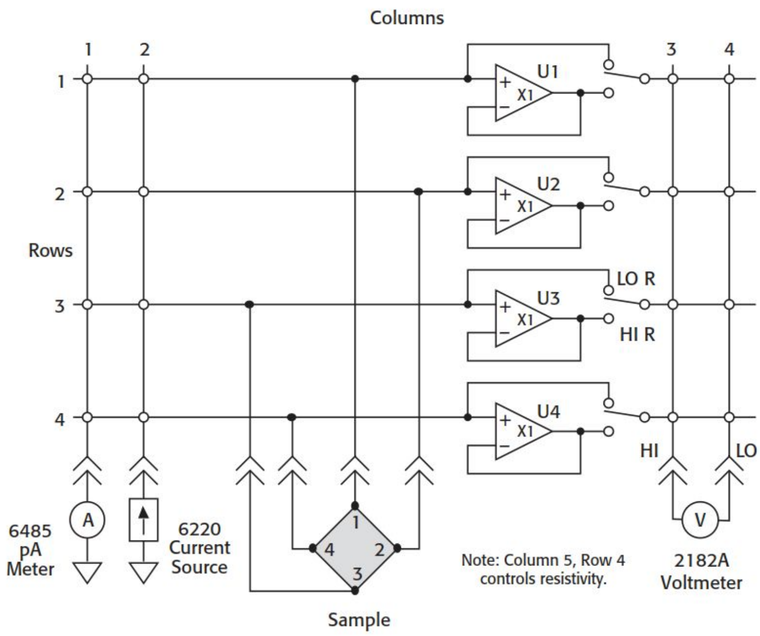



Figure 3.5.: Typical van der Pauw measurement setup by use of a Keithley 7065 Hall effect card [6].

For a complete van der Pauw resistivity and Hall effect determination, twelve measurements need to be taken. If the measurement devices are able to reverse polarity, still six measurements require rewiring of the specimen contacts. This can either be done manually or automated by use of a switch matrix (see figure 3.5). Another method is the use of four Source Measurement Units (for example two two-channel Keithley SourceMeters) in the configuration shown in figure 3.6. The benefit of using four Source Measure Units instead of a switch matrix is, that the sample can be directly wired to the Source Measure Units and no other circuitry is needed. The Source Measure Units have an electrometer grade high impedance input and therefore also samples with low conductivity can be measured accurately. An implementation of the above described van der Pauw measurement method using two two-channel Keithley SourceMeters has been implemented in Python and can be found in appendix A.2.2.

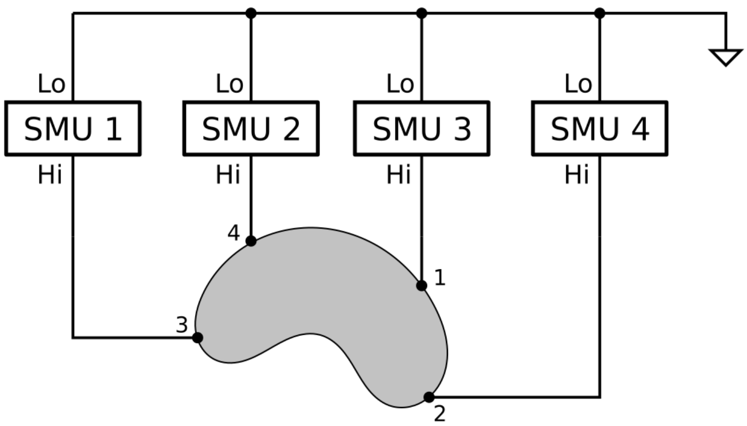



Figure 3.6.: Van der Pauw measurement configuration to measure an arbitrary shape with four Source Measure Units eliminating the need for a switch matrix.

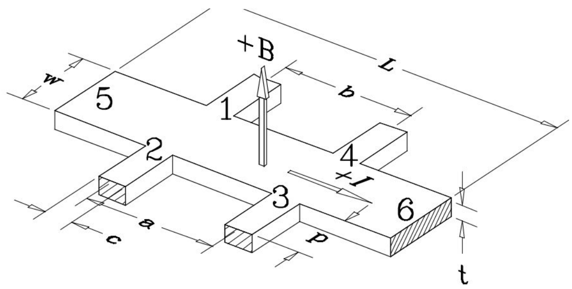



Figure 3.7.: Hall bar geometry with contacts labeled from 1 to 6 and indicated current and magnetic field direction [12].

## 3.2. Hall bar type

Hall bars (as shown in figure 3.7) approximate the ideal geometry for Hall effect measurements. A constant current density flows along the long axis of the specimen, perpendicular to an external magnetic field. The Hall voltage can be measured across contact pairs that are placed symmetrically along the long axis of the sample.

In figure 3.7 a current is driven across contacts 5 and 6 ($I_{56}$). Resistivity can be measured by measuring the voltage drop across contacts 2 and 3 ($U_{23}$) and across contacts 1 and 4 ($U_{14}$). The use of separate contacts for resistivity measurement is due to the principle of four contact resistivity measurement. It ensures that contact resistance can be neglected.

The resistivity of the specimen can be calculated by the measured current and voltage and the known geometry dimensions as shown in figure 3.7:

$$\rho = \frac{U_{23}}{I_{56}} \cdot \frac{w\,t}{a} \tag{3.8}$$

To get a higher accuracy, the polarity of the current should be reversed and additionally the voltage drop across contacts 1 and 4 should be measured. With those four measurements the mean resistivity can be calculated:

$$\rho_A = \frac{U_{23}^{+} - U_{23}^{-}}{I_{56}^{+} - I_{56}^{-}} \cdot \frac{w\,t}{a} \tag{3.9}$$

$$\rho_B = \frac{U_{14}^{+} - U_{14}^{-}}{I_{56}^{+} - I_{56}^{-}} \cdot \frac{w\,t}{b} \tag{3.10}$$

$$\rho = \frac{\rho_A + \rho_B}{2} \tag{3.11}$$

According to ASTM F76-08 standard $\rho_A$ and $\rho_B$ need to be equal within $\pm 10\,\%$, otherwise the specimen is too inhomogeneous and a more uniform specimen is required [13].

The Hall coefficient can be obtained by measuring the Hall voltage between contacts 1 and 2 when a magnetic field is applied.

$$R_H = \frac{t}{B} \cdot \frac{U_{12}}{I_{56}} \tag{3.12}$$

To further enhance accuracy and eliminate the influence of geometrical errors, it is advised to perform multiple measurements with reversed current polarity, reversed magnetic field by 180° and also measurements of the Hall voltage across the contact pair 3 and 4. $B^-$ indicating a 180° rotated magnetic field in respect to $B^+$. The rotation of the magnetic field can either be achieved by physical rotation of the magnet 180° or, if an electromagnet is used, by reversing the current that is flowing through the coils of the magnet. With these additional measurements, the Hall coefficient can be calculated as:

$$R_{HA} = \frac{U_{21}^+(B^+) - U_{21}^-(B^+) + U_{21}^+(B^-) - U_{21}^-(B^-)}{I_{56}^+(B^+) - I_{56}^-(B^+) + I_{56}^+(B^-) - I_{56}^-(B^-)} \cdot \frac{t}{B} \tag{3.13}$$

$$R_{HB} = \frac{U_{34}^+(B^+) - U_{34}^-(B^+) + U_{34}^+(B^-) - U_{34}^-(B^-)}{I_{56}^+(B^+) - I_{56}^-(B^+) + I_{56}^+(B^-) - I_{56}^-(B^-)} \cdot \frac{t}{B} \tag{3.14}$$

If the values for $R_{HA}$ and $R_{HB}$ do agree within ±10 %, the Hall coefficient $R_H$ can be calculated as follows. If the deviation is higher, a more uniform sample is needed.

$$R_H = \frac{R_{HA} + R_{HB}}{2} \tag{3.15}$$

For materials with dominant carrier type, the carrier density ($n$) and the carrier mobility ($\mu_H$) can be calculated using equations 2.12 and 2.13, as described in chapter 2.2.

## 3.3. Low level current measurement

When working with low conductive samples, it is necessary to pay closer attention to possible sources and sinks of unwanted currents. One of the major error sources are leakage currents and parasitic capacities within the measurement setup and the measurement devices itself.

Leakage currents are currents across stray resistance paths that bypass the device under test (DUT). Those stray resistance paths are most often currents across insulators. For example the leakage current $I_L$

across the cable insulation resistance $R_L$ as shown in figure 3.9(a). Additionally also parasitic capacitance need to be charged when an applied or measured voltage changes. In contrast to leakage currents, the currents caused by parasitic capacitance will diminish over time.

To investigate the performance of a measurement setup, a step function test, as shown in figure 3.8, can be performed. To run this test, a voltage source and a current meter is needed. A Source Measure Unit is a measurement device that has both of those instruments build into the same device. The following exemplary description will use a Source Measure Unit but is adaptable to any kind of source measure arrangement of instruments.

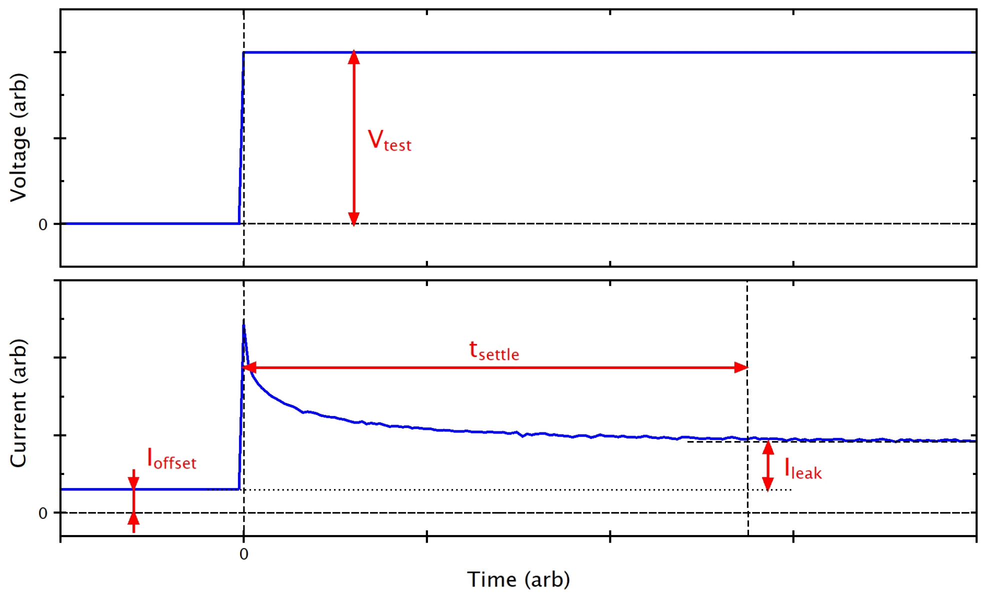



Figure 3.8.: Step function test to obtain information about system settling time and system leakage current. The top graph shows the test voltage increasing as a step function. The bottom graph shows the current change caused by the change of the voltage.

The procedure of this test is as follows:

1. The specimen is disconnected form the devices. The cables need to stay attached.
2. Initially the test voltage is set to zero $V_{test} = 0$ and the current is recorded. This is the offset current $I_{offset}$ of the system.
3. The test voltage $V_{test}$ is then instantly increased to $V_{test} > 0$ (step function as seen in the top figure 3.8) and the trend of the current $I(t)$ is recorded.

4. The current $I(t)$ will converge to a value that represents the leakage current $I_{leak}$. The time it takes the system to converge is the settling time $t_{settle}$ of the system.

To gain correct measurement values of a device under test, a measurement must be taken after $t_{settle}$ and the offset current needs to be subtracted. To distinguish between internal and external currents, the test can be as well done with no cables attached to the measurement device. The results then represent the internal leakage currents.

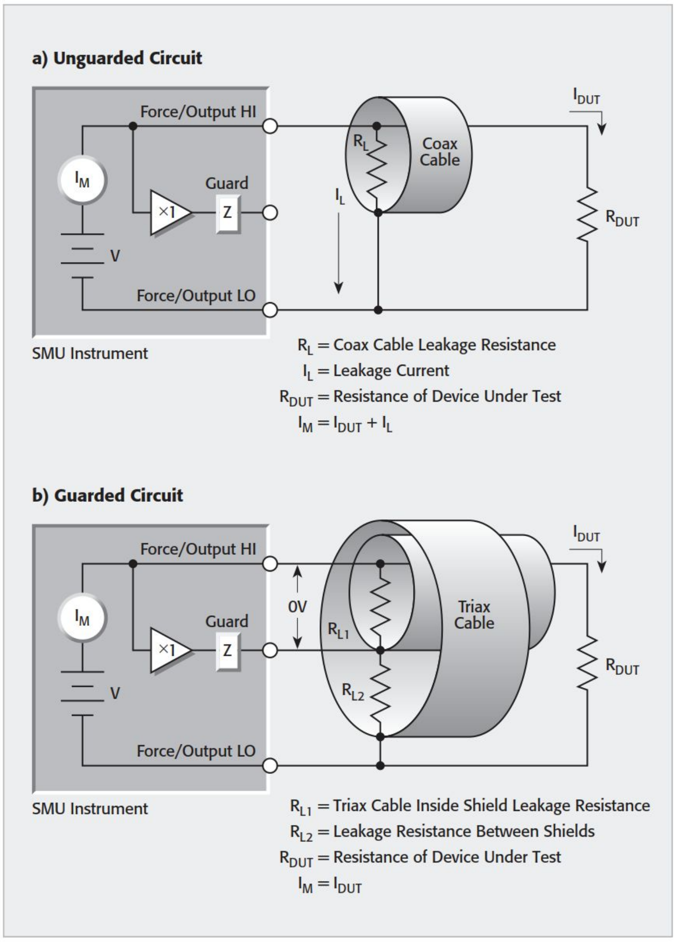



Figure 3.9.: Guarding the leakage resistance of a cable with an SMU instrument [19].

To avoid these unwanted currents, guarding can be used as shown in figure 3.9(b). Therefore the inner conductor of the wire is surrounded by an additional conductor (triaxial cable) whose voltage is set to the same level as the voltage level of the inner conductor. This way there is no potential difference between the inner conductor and the next surrounding conductor and therefore no current is flowing through $R_{L1}$.

The guard itself is driven by a dedicated circuit and current flowing from guard to the outer ground shield (indicated by $R_{L2}$) does not add to the measured current.

The concept of guarding can be applied to both AC and DC measurements. However on AC it must be taken into consideration that parasitic capacities need to be charged and discharged from the guarding circuit. The period of the AC signal should be larger then the settling time evaluated with the step function test.

Guarding will decrease the leakage current $I_L$ and more importantly reduce the settling time of the system enabling for faster measurement intervals.

## 3.4. Low level voltage measurement

Hall voltages can be in the μV range and below, therefore making these measurements specially sensitive to thermal voltages on connections, electrochemical potentials and electromagnetic interference. To reliably measures such small signals, the lock-in measurement technique can be used. The lock-in technique uses AC signals, therefore the specimen as well as the setup must be designed to deal with AC signals. Special attention needs to be paid to ohmic contacts to the sample.

A lock-in amplifier is a frequency and phase sensitive measurement device. Only signals with the same frequency and phase as a reference frequency are amplified and contribute to the output signal. The best instruments manage to recover signals that are one million times smaller than the noise present [20].

The working principle of a lock-in amplifier is, that the input signal (index sig) is multiplied with the provided reference signal (index ref) and integrated in a low pass filter. The cross correlation of signals with different frequency is zero and they will cancel out during the integration time.

The experimental setup needs to ensure that the AC signal source is fed to the lock-in amplifier in two ways as shown in figure 3.10. This can either be achieved by use of the internal AC reference (that most lock-in amplifiers have) or by using the TTL (logic level) output of the external signal generator that is used for the experiment.

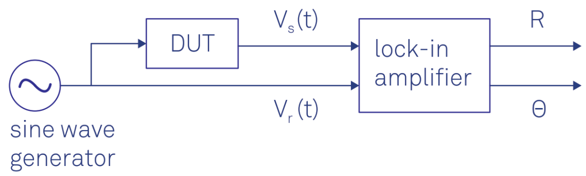



Figure 3.10.: Experimental setup of a lock-in amplifier with external AC source providing the device under test (DUT) test voltage as well as the reference signal [20].

Mathematically this is a multiplication of two waves with amplitude $V$, frequency $\omega$ and phase $\theta$. $V_{psd}$ represents the output voltage of the phase sensitive detection device.

$$V_{psd} = V_{sig}\sin(\omega_{sig}t + \theta_{sig}) \cdot V_{ref}\sin(\omega_{ref}t + \theta_{ref}) \tag{3.16}$$

which can be rewritten as:

$$\begin{aligned} V_{psd} = & \frac{1}{2} \cdot V_{sig}V_{ref}\cos\left((\omega_{sig} - \omega_{ref})t + \theta_{sig} - \theta_{ref}\right) - \\ & \frac{1}{2} \cdot V_{sig}V_{ref}\cos\left((\omega_{sig} + \omega_{ref})t + \theta_{sig} + \theta_{ref}\right) \end{aligned} \tag{3.17}$$

After the multiplication the signal is filtered in a low pass filter eliminating all the AC content. The output signal is then a DC signal that is proportional to the input signal amplitude $V_{sig}$ as shown in equation 3.18 [21].

$$V_{psd} = \frac{1}{2} \cdot V_{sig}V_{ref}\cos\left(\theta_{sig} - \theta_{ref}\right) \tag{3.18}$$

When choosing the frequency for the measurement the following should be considered:

- The higher the frequency, the more parasitic capacities will influence the measurement.
- The lower the frequency, the longer the integration time must be for the lock-in to output a correct signal.
- Be aware that other measurement equipment (for example a multimeter) has a minimum frequency for AC signals to be measured correctly. Consult the devices manual for detailed information about the AC range.

- Avoid frequencies with high content of noise. For example the mains frequency (50 Hz) and multiples of this frequency should be avoided.

When using a lock-in setup and only measuring with AC, the information about the polarity of the Hall voltage is within the phase output of the lock-in. A phase close to zero indicates a positive Hall voltage whereas a phase of approximately 180 ° indicates a negative Hall voltage. A more reliable method is to use simultaneous AC and DC measurement as shown in chapter 4.3.1.

# 4. Experimental Section

## 4.1. Sample Preparation

In order to verify the performance of the new Hall measurement laboratory, standardized Hall samples were needed. Neither the National Institute of Standards and Technology (NIST) nor commercial suppliers offer thin film Hall standards that could be used to calibrate and verify the setup against.

The Russian Academy of Sciences, Ioffe Physical Technical Institute (Russia, St. Petersburg) is specialized on Hall measurements on highly doped semiconductors and offered to measure provided samples to compare results. Therefore custom standard samples were needed, that can be measured at the Ioffe Institute and at the new Hall measurement laboratory at the Institute of Solid State Physics, TUGraz. The material chosen for the standard test samples were ITO because of its well known properties as well as the commercially available structuring possibilities.

The ITO base material was ordered from Sigma-Aldrich in the form of Indium Tin Oxide coated rectangular glass slides[1] that were laser structured. The laser structuring process produces geometries with an error of only $\pm 2\,\mu\text{m}$ and is therefore ideally suited to produce standard Hall geometries.

For the purchased ITO the following properties were listed by Sigma Aldrich:

- thickness 600 to 1000 Å
- surface resistivity 15 to 25 $\Omega$/sq
- transmittance $> 78\,\%$

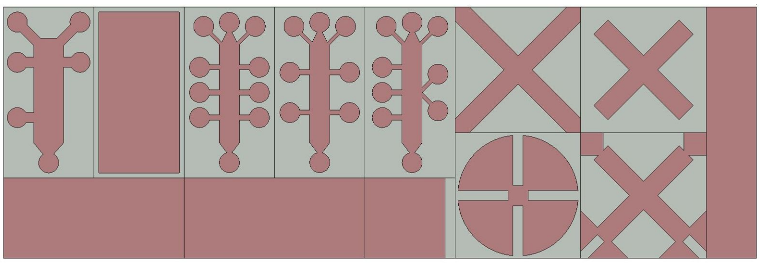


Figure 4.1.: Overview of the laser structured Indium Tin Oxide Hall geometries. Indium Tin Oxide is shown in red color, glass substrate is shown in gray. Left: different Hall bar geometries. Right: Van der Pauw Hall geometries.

### 4.1.1. Geometry Considerations

With laser structuring, it is possible to produce arbitrary geometries. The design of the samples, that were produced, is based on the recommended geometries as stated in the ASTM F76-08 standard [13] with specifications as listed in table 2.2 and table 2.3. In figure 4.1 a overview of the produced geometries is shown. On the left part Hall bar geometries have been structured and on the right part van der Pauw geometries were structured.

The laser structuring was performed by the company LaserMicronics GmbH (Garbsen, Germany). The laser structuring works by evaporating ITO from the surface where it is not needed leaving the remaining surface covered with ITO. The benefit of this method is the fact, that the ITO surface is not brought into contact with any other material (as it would have been when using photolithography as structuring method).

In total six slides as shown in figure 4.1 were manufactured. From two slides selected geometries were sent to the Ioffe Institute to measure the ITO samples with their calibrated equipment. Results and comparison with measurements done at the Institute of Solid State Physics will be presented and discussed in chapter 5.

The Hall bar geometries differ in the aspect of having more than two contacts along the long axis of the sample. In figure 4.2(a) a sample as listed in the ASTM standard is shown. Three of the six side contacts are grayed out because they are only needed to enhance accuracy on non ideal samples by providing the possibility of performing multiple measurements. The Hall voltage as well as resistivity can be measured with the remaining five contacts.

[1] http://www.sigmaaldrich.com/catalog/product/aldrich/636916

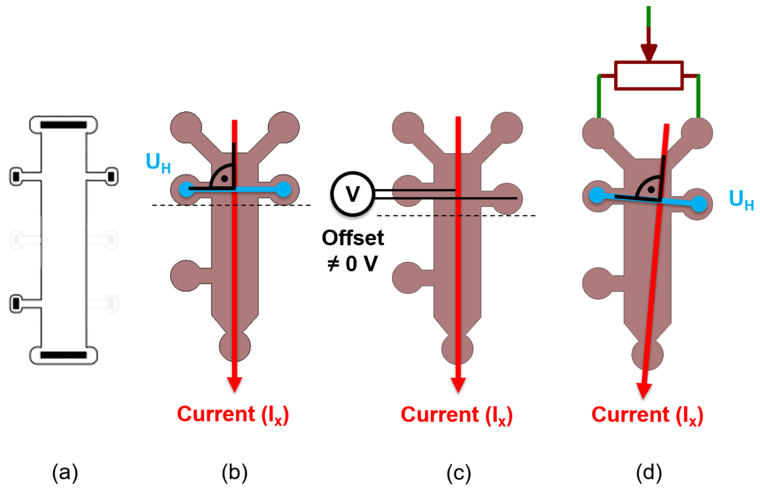



Figure 4.2.: Modification of the recommended Hall bar shape in order to compensate for geometrical errors by adjustment of the current path through the specimen. (a) recommended structure by ASTM, (b) ideal structure with no geometrical offset, (c) geometrical offset of the Hall contacts, (d) adjustment of the current path.

If samples have ideal geometry without any error the current path is exactly along the long axis and the Hall voltage can be measured perpendicular as shown in 4.2(b). If however there is even a small geometrical error, as shown in 4.2(c), a voltage drop across the contact will occur without magnetic field applied. This offset is most likely much bigger than the Hall voltage. Even if the offset is known (and can be subtracted from the measurement), it causes a reduction of the usable dynamic range of measurement devices. If the offset is too large, it is maybe not even possible to measure the Hall effect because the resolution of measurement devices is too small.

In order to circumvent this problem, the current path inside the sample can be modified as shown in 4.2(d). A variable resistor is connected to the top two contacts. By changing the ratio of resistance the current distributes unevenly across the two contacts resulting in a tilted mean current path. Now the Hall voltage can be once again measured exactly perpendicular to the current path and no offset voltage is present. To adjust the potentiometer to the correct position, a current is driven through the sample and the potentiometer is adjusted to a position where the offset voltage is zero. This offset correction has to be done when no magnetic field is applied.

The value of the potentiometer should be chosen carefully. To small values will have not the desired effect as to large values will reduce the maximum possible current to be driven through the sample. It should be also considered that larger values of the potentiometer have the benefit that a change in sample contact resistance during measurement has less influence on the resistance ratio and therefor the current path will not be changed significantly.

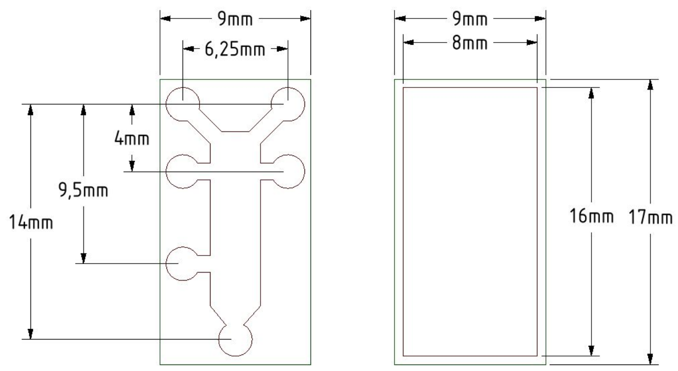



Figure 4.3.: Dimensions of selected ITO Hall geometries. Left: Hall bar geometry used at the Ioffe Institute, Russia. Right: Rectangle sample that can either be measured with the van der Pauw method or Hall bar method.

In figure 4.3 the two Hall bar structures are shown that were selected for initial measurements and shipped to the Ioffe Institute. The geometry on the left is a custom design as used by Ioffe Institute fitting in their specialized test fixture. The geometry on the right is a rectangular shape that can either be measured with the van der Pauw method or Hall bar method.

### 4.1.2. Sample Investigation

For calculation of the resistivity, the carrier density and the carrier mobility, physical dimensions of the specimens are needed. The lateral dimensions are (based on the laser structuring process) known to $\pm 2\,\mu m$ and listed in figure 4.3.

The thickness was measured with spectroscopic ellipsometry (wavelength range of 371 to 1000 nm with ellipsometer M-2000V, J.A. Woollam Co.Inc.). The measured angles as well as the fitting model is shown in figure 4.4. The purchased samples have an additional $SiO_2$ layer between the glass substrate and the ITO layer. This $SiO_2$ layer has also been included in the fitting model.

The following parameters have been evaluated:

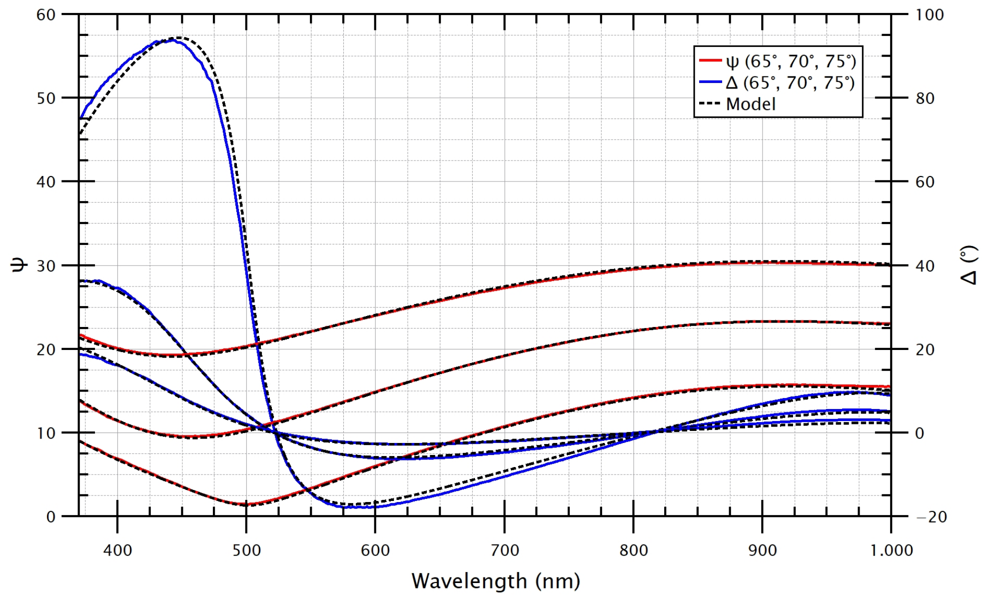



Figure 4.4.: Spectroscopic ellipsometry measurement of ITO specimen at different incident angles of 65 °, 70 ° and 75 ° with overlayed fitting model [2].

- Thickness $SiO_2$: $(23 \pm 2)$ nm
- Thickness ITO: $(78 \pm 2)$ nm
- refractive index ITO (@632.8 nm): 1.75

The ITO was investigated with spectroscopic ellipsometry before and after laser structuring. No significant changes of material properties were found, confirming that the laser structuring process did not alter the ITO base material.

## 4.2. Laboratory Setup

### 4.2.1. Magnet Characterization

The magnet used in the setup of the new magnetic laboratory is a Bruker type B-E15 B8 that was build prior to 1980 (shown in figure 4.5). The manufacturer was not able to provide any specifications for the magnet so it was necessary to ensure that the homogeneity of the magnetic field is sufficient at the pole distance needed for the closed cycle cryostat.

[2]Measurement and fitting model done by Alberto Perrotta

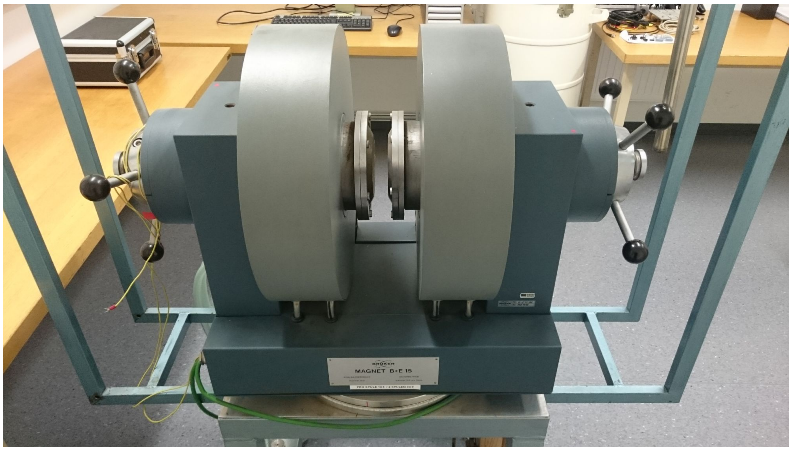


Figure 4.5.: Bruker Magnet type B-E15 B8 used for the new magnetic laboratory shown on the old support frame prior to operation.

To evaluate the homogeneity of the magnetic field, a test setup (figure 4.6) was build to move a Hall sensor element in two dimensions across the area between the pole shoes. The mechanical construction of the 2D-stage was made with MakerBeam 10 x 10 mm T-slot aluminum profiles. The movement of the stage was driven by two NEMA 17 stepper motors connected to a PepRap Arduino Mega Pololu Shield with A4988 stepper motor drivers. The basic control of speed, acceleration and position was done with an Arduino Mega2560 using the AccelStepper library[3]. The Arduino source code of the stepper control can be found in appendix A.2.3. The Arduino itself was connected to the computer and movement commands and measurement was controlled by a Python program. Figure 4.7 shows such a two dimensional scan at a pole distance of 4.5 cm.

The Hall sensor used was a ChenYang CYSJ362A GaAs Hall element[4] with an Hall output voltage of 2 mV/mT ($I_{sensor} = 5$ mA). The measurement of the Hall voltage was done with a Keithley 2000 multimeter.

In figure 4.8 a cross sectional scan horizontally through the center of the pole shoes is shown at different magnetic fields. The magnetic field inside the sample area varies only 0.1 % independent of the applied magnetic field.

During the setup of the laboratory the magnet needed to be lifted to a new support frame. To estimate the weight of the magnet it has been

[3] http://www.airspayce.com/mikem/arduino/AccelStepper/

[4] http://www.sonnecy-shop.com/en/linear-hall-effect-sensors-elements-cysj362a-max.-sensitivity-3.1-4.1-mv/mt-measuring-range-3t.html

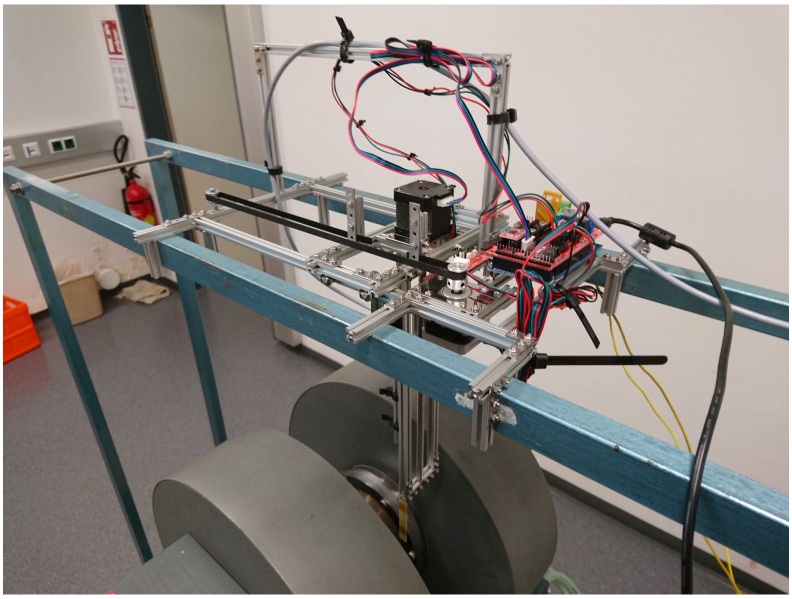


Figure 4.6.: 2D-stage for automated two dimensional movement of a Hall sensor element between the pole shoes of the Bruker electromagnet.

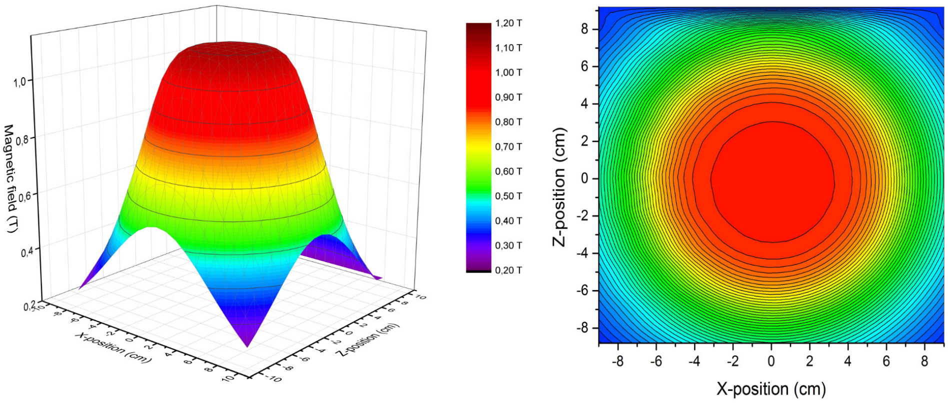



Figure 4.7.: 2D measurement of the homogeneity of the magnetic field measured with a Hall sensor element between the pole shoes of the Bruker magnet at a pole distance of $d_{poles} = 4.5\,cm$.

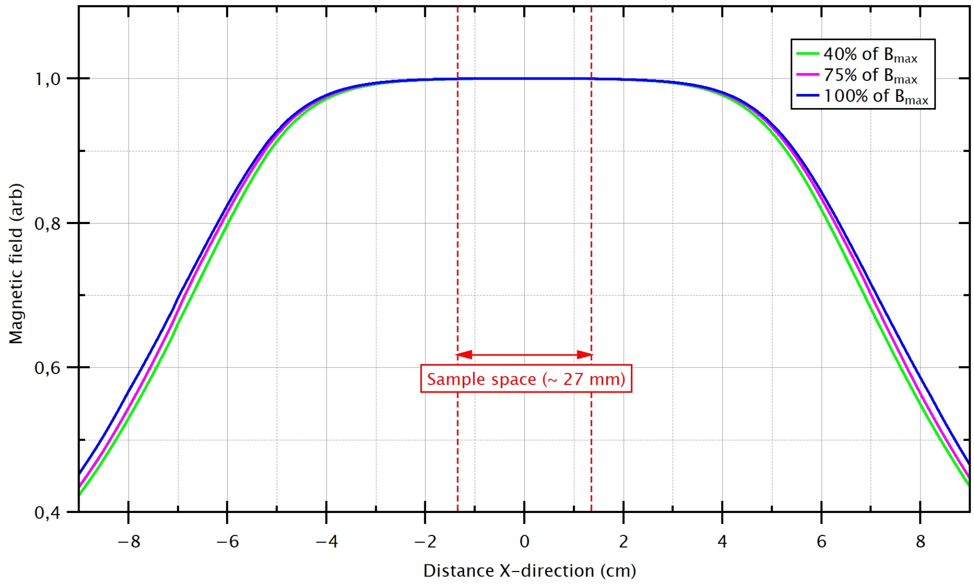



Figure 4.8.: 1D measurement between the pole shoes of the magnet at different magnetic fields. The Hall senor element was moved horizontally through the center of the circular pole shoes at a pole distance of $d_{poles} = 4.5\,\mathrm{cm}$.

modeled in SolidWorks with the appropriate material parameters for the different parts of the magnet. The estimate weight evaluated by the SolidWorks model is approximately 650 kg. On top of the magnet two M16 screw threads can be used to attach lifting equipment.

### 4.2.2. Closed Cycle Cryostat

To perform temperature dependent measurements, a closed cycle cryostat has been custom made for narrow gap magnetic applications by the company Advanced Research Systems (ARS), USA.

The basic operation principle of a closed cycle cryostat is, that a compressor system supplies compressed helium to the cold head of the cryostat through gas lines. The gas expands in the cold head to provide refrigeration, by expanding the high-pressure helium to low pressure, and then returns to the compressor to be compressed again. The helium cycle is completely closed and no liquid helium is needed for operation.

A sample holder is attached at the tip of the cold head by screwing the sample holder into the 800 K high temperature interface. Due to manufacturing processes, the rotation of the sample holder can vary. In order to achieve a horizontal sample orientation when the cold head

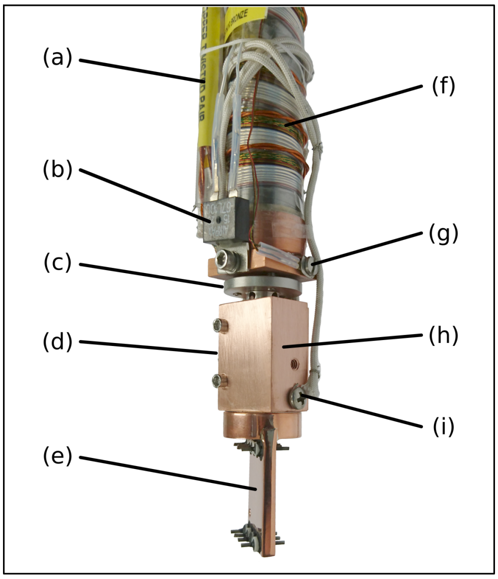



Figure 4.9.: Cold head of the ARS CS202AE/800K-DMX.3-1AL closed cycle cryostat. (a) free length Cernox temperature sensor, (b) over temperature thermostat, (c) sapphire thermal insulation interface, (d) PTR temperature sensor for high temperatures, (e) sample mounting space, (f) second cold stage, (g) diode reference temperature sensor, (h) 800 K interface, (i) thermocouple temperature sensor.

is tilted out of the magnet, silver gaskets can be used to act as spacer between the cold head and the sample holder. The silver gaskets also improve the thermal contact between the high temperature interface and the sample holder.

The cryostat is capable of reaching a base temperature of 8 K in less than 70 min. The exact cool down characteristic can be seen in figure 4.10. The cryostat is additionally equipped with a heating interface that can go up to 800 K. Refer to figure 4.9 for detailed description of the construction of the cold head and the position of the four different temperature sensors.

For very sensitive measurements it is required to minimize sources of interference (such as the heating controller). The best way to perform such measurements is, to turn off every unneeded devices and perform measurements during the self heat-up of the system. In figure 4.11 the temperature trend is shown during self heat-up (the compressor is turned off and no controlled heat-up is initiated). The curve shown in figure 4.11 represents the heat-up when no power is brought into the system by measurements on a device under test. However little power is brought into the system by the constant measurement of the

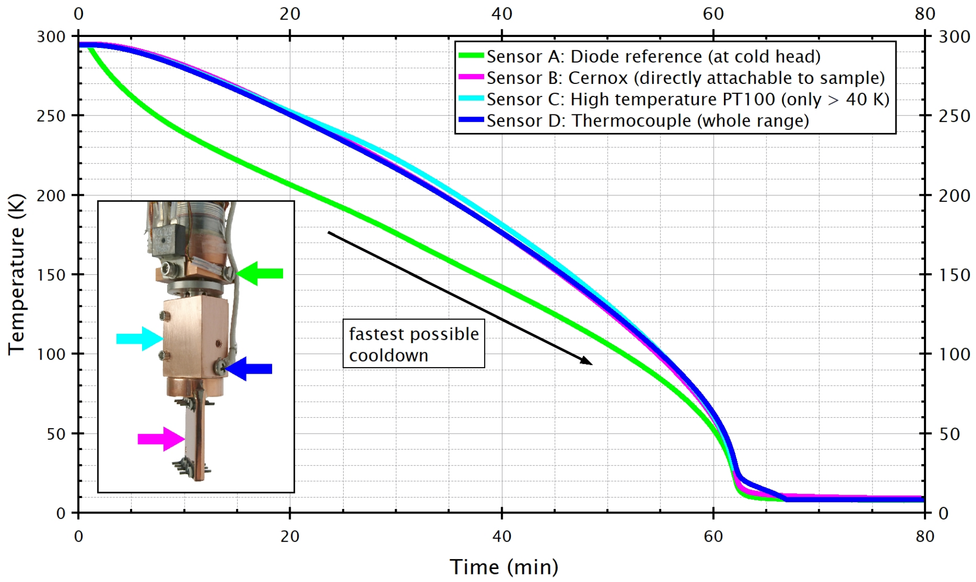



Figure 4.10.: Cool down characteristic of the Advanced Research Systems Closed Cycle Cryostat CS202AE-DMX-3-1AL cold head.

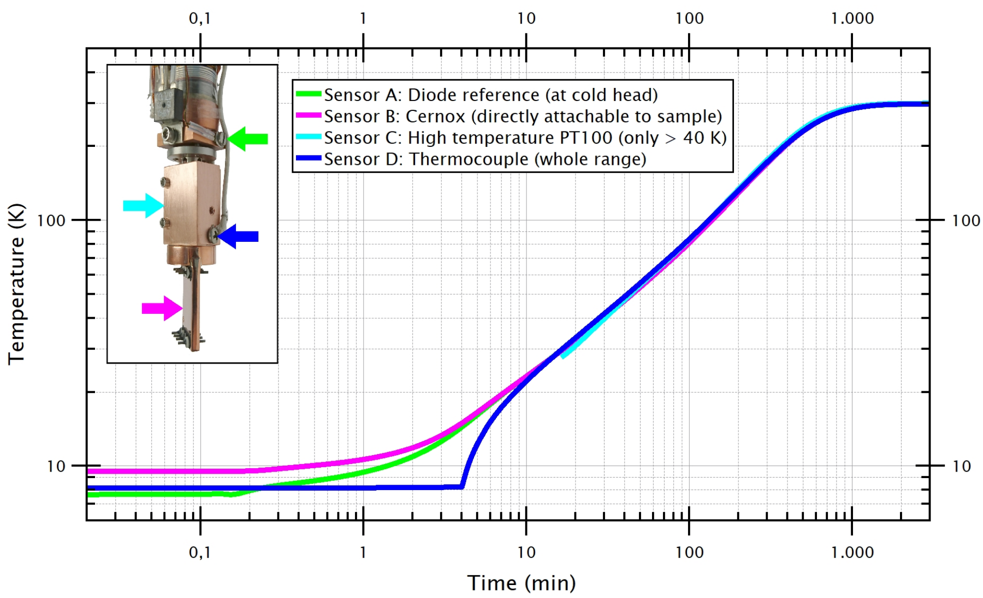



Figure 4.11.: Self heat up characteristic of the Advanced Research Systems Closed Cycle Cryostat CS202AE-DMX-3-1AL cold head from base temperature to room temperature.

temperature. The heat-up time from base temperature (8 K) to room temperature 297 K is longer than 1000 min. The thermal insulation vacuum during this test was constant at $10^{-6}$ mbar.

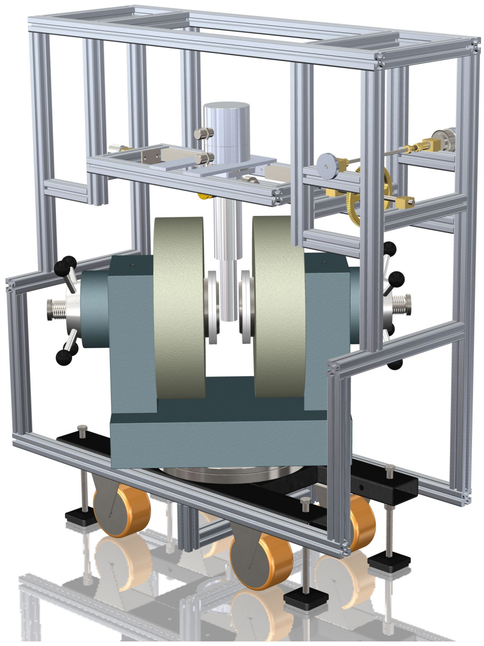


Figure 4.12.: SolidWorks rendering of the support frame holding the closed cycle cryostat between the pole shoes of the magnet [5].

To mount the cold head, a support frame has been build using 40 x 40 mm non magnetic ITEM aluminum profiles. The profiles enable for accurate position adjustment and provide flexibility to extend or modify the setup. The support frame, the cold head and the magnet have been 3D-modeled in SolidWorks prior to construction to ensure optimal interaction of the different devices. A rendering of the model is shown in figure 4.12.

The cold head is mounted in a rotate-able frame that can be tilted by 90 $^\circ$ as shown in figure 4.13. When the cold head is tilted horizontally, easy access to the sample is provided by removal of the cold heads vacuum shroud and radiation shield. When the cold head is tilted vertically, the geometry of the support frame ensures that the sample resides in the center of the magnet pole shoes.

For thermal insulation a pressure of $p < 10^{-4}$ mbar or lower is needed. This setup uses a Pfeiffer vacuum pumping stand consistent of a dry membrane pump combined with a turbomolecular pump 4.13(g).

[5]SolidWorks construction and rendering done by Martin Kornschober.

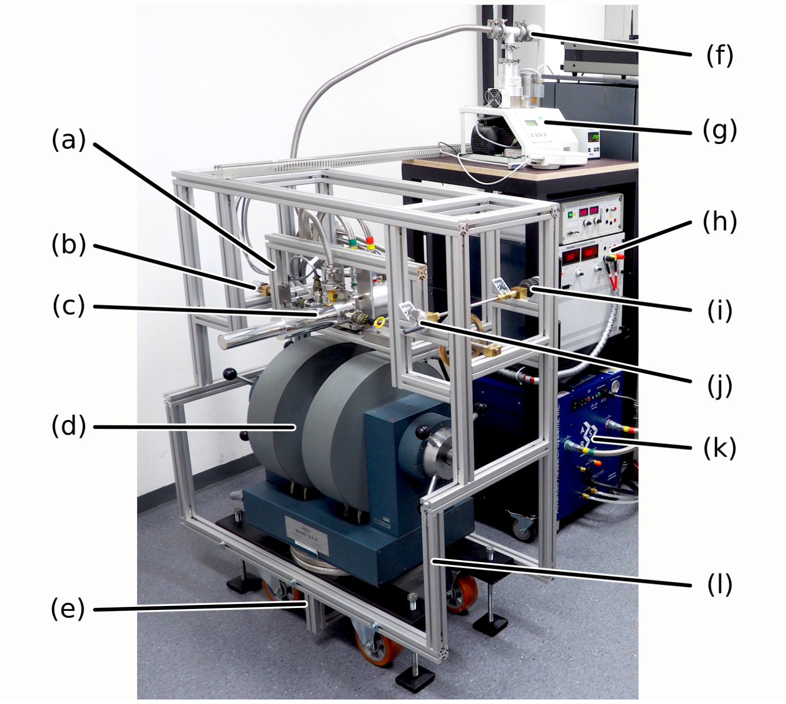



Figure 4.13.: Support frame for the closed cycle cryostat. (a) rotate-able frame to tilt the cold head for sample access, (b) preparation for absolute position control of the cold head, (c) rotate-able magnet, (e) preparation for motorized movement and absolute position control of the magnet, (f) pressure sensor, (g) vacuum pump system consisting of a dry membrane pump and a turbomolecular pump, (h) magnet power supply, (i) preparation for motorized movement of the cold head, (j) hand wheel for manual tilting of the cold head, (k) Helium compressor, (l) ITEM aluminum profiles.

The cryostat is attached by use of flexible bellows to minimize the transmission of vibrations from the vacuum stand to the cold head.

The temperature sensors are connected to a LakeShore 336 temperature controller (figure 4.14) that has four sensor inputs (Senor A, B, C, D).


Figure 4.14.: LakeShore 336 temperature controller shown with disabled heating and cryostat at room temperature.

### Sensor A

*Type:* LakeShore DT-670B-SD diode temperature sensor.
*Range:* From 1.4 K to 500 K.
*Position:* Reference sensor mounted at the second cold stage 4.9(f) above the thermal insulation interface 4.9(c).
*Remark:* High magnetic field-induced error.

### Sensor B

*Type:* LakeShore Cernox CX-1030-SD-HT-1.4M thin film resistance temperature sensor (S/N-X119992).
*Range:* Calibrated within 1.4 K to 420 K; most accurate sensor for low temperatures.
*Position:* Can be positioned freely inside the cryostat; usually placed near specimen 4.9(a).
*Remark:* Low magnetic field-induced error.

### Sensor C

*Type:* LakeShore PT-103 platinum resistance sensor.
*Range:* From 30 K to 800 K.
*Position:* Mounted on the side of the heating stage 4.9(d).
*Remark:* Used for better accuracy at high temperatures. The PT-103 package is a special non-magnetic variant of a PT-100 temperature sensor.

### Sensor D

*Type:* Thermocouple Type E (Chromel-Constantan).
*Range:* From 3.15 K to 953 K.
*Position:* Mounted on the heating stage close to the sample holder 4.9(f).

*Remark:* Covers the whole temperature range of the cryostat and is used for PID control. For magnetic fields $B < 1\,T$ the error is $\Delta T/T < 1\,\%$.

The temperature sensors have been calibrated by Advanced Research Systems and the calibration curves are stored in the internal memory of the LakeShore 336 temperature controller.

The LakeShore 336 temperature controller can be controlled via computer interface. In this setup a LAN connection was used. A library for easy operation was build using Python with pyVISA (details can be found in chapter 4.2.3).

### 4.2.3. Programming of measurement devices

The system used to control the measurement devices is shown in figure 4.15 and consists of the physical connection to the instrument, the appropriate software drivers, the National Instruments Virtual Instrument Software Architecture (NI-VISA) and the Python package PyVISA to interface with the NI-VISA interface. This ensures that the devices can be accessed from within Python.

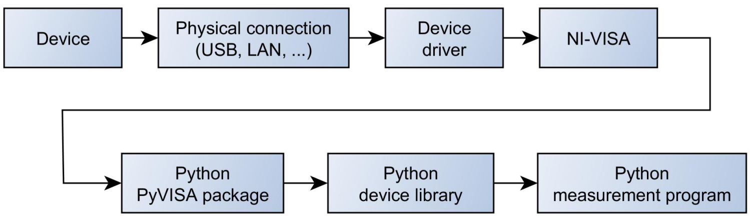



Figure 4.15.: Structure of the involved components to control a measurement device using Python.

The Virtual Instrument Software Architecture (VISA) is a standard for configuring, programming, and troubleshooting instrumentation systems comprising GPIB, VXI, PXI, Serial, Ethernet, and/or USB interfaces[6].

For an easy and safe operation, for each device a custom Python library has been programmed that provides functions for the user to interact with the device. The measurement program itself uses those libraries which ensures that the source code stays simple to read and

[6] https://www.ni.com/visa/

maintain. The design payed special attention to keeping code readable so that future operators (that are new to Python) can understand the programs and operate devices instantly. The Python implementation uses Python 3.x and is not compatible to Python 2.x.

The documentation of the libraries is contained within the library in the form of standardized docstrings[7]. This ensures that documentation is consistent with the functionality of the library and can be accessed during programming by any modern development environment.

When using a measurement devices within Python, the first thing that needs to be done is to connect to the device. Upon successful connection, the device with its implemented functions can be accessed by the returned device object. Such an initialization procedure is nearly the same for every device and looks like the following example code.

```
from libs.DeviceManufacturer import ModelXYZ

# connect to the device over the interface as listed in the NI-VISA manager
# this example uses a GPIB connection
device = ModelXYZ()
device.connect("GPIB0::2::INSTR")

# (optional)
# Enable debug output so we see the commands that are
# sent to and received from the device
device.enable_debug_output()

# Reset the device to defaults
device.reset()

#
# ## Device is ready for operation and can be used. ##
#

# disconnect from the device
device.disconnect()
```

The variable *device* should be replaced with a meaningful name that describes which device is accessed. This makes code easier to maintain within the program.

In the following section some basic programming examples are listed for the devices used in the following experiments.

### Switch matrix

The switch matrix is used to connect different measurement devices to the sample and is specially needed if van der Pauw measurements are

[7] https://www.python.org/dev/peps/pep-0257/

done. The Agilent 3499A switch controller can be equipped with up to five switch cards with different functionalities. The switch cards have IDs related to the installed slot. The first switch card has ID 100 and the relays on that card can be accessed by sub-IDs.

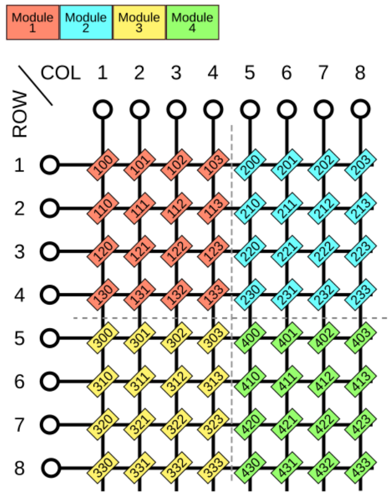



Figure 4.16.: Switch matrix configuration using four Agilent 44473A 4 x 4 2-wire switch modules that are connected to obtain one 8 x 8 matrix configuration.

This setup uses a special configuration in which four 44473A 4 x 4 2-wire switch modules are connected to obtain one 8 x 8 matrix configuration. To provide easier usage, the Python library was extended to support this special configuration as shown in figure 4.16. The source code of the Python library can be found in appendix A.2.4. To operate the switch matrix the following commands can be used:

```
# Close connection row 6 to column 3 then
# wait for 5 seconds and open the connection again
switch.close_matrix(6, 3)
sleep(5)
switch.open_matrix(6, 3)
```

To directly control a relay of the switch controller, the following commands can be used. The ID of the corresponding relay can be seen in figure 4.16 at the crossing of the wires that should be connected.

```
# Close connection row 6 to column 3
# This is the equivalent of switch.close_matrix(6, 3)
# wait for 5 seconds and open the connection again
switch.close_channel(312)
sleep(5)
switch.open_channel(312)
```

### Lock-in amplifier

A detailed description of the operation principle of lock-in amplifiers can be found in chapter 3.4.

In this setup two different lock-in amplifiers were used. A Stanford Research Systems SR830 dual phase lock-in and the Princeton Applied Research Model 5210 dual phase lock-in. For both devices a Python library was programmed that can be found in appendix A.2.5 and A.2.6

After connecting to the device (as shown in the pseudo code in chapter 4.2.3) the basic measurement parameters need to be configured:

```
# use the internal signal generator as reference clock
# with an output level of 0.1 V at a frequency of 11 Hz
sr830.use_internal_reference()
sr830.set_reference_frequency(11)
sr830.set_sine_output_level(0.1)

# enable the 2 line filters (50 Hz and 100 Hz)
sr830.enable_line_filters()

# set the input to differential mode
sr830.set_input_mode_A_minus_B()
sr830.set_input_shield_to_ground()
sr830.set_input_coupling_ac()

# set the time constant and the filter
sr830.set_time_constant(0.1)
sr830.set_filter_slope(18)

# set the reserve
sr830.set_reserve_low_noise()

# set the sensitivity to 50mV
sr830.set_sensitivity(50E-3)

# set the displays to show real part and the phase shift
sr830.display_ch1_r()
sr830.display_ch2_phi()
```

When the lock-in is configured properly, measurements can be taken. To command the lock-in to take a measurement the following commands can be used:

```
r = sr830.read_r()
phi = sr830.read_phi()
```

### Magnet power supply

The power supply used to drive current through the magnet is a Heinzinger PTN40-125. The Python library to control the power supply can be found in appendix A.2.7.

To control the magnetic field, the power supply is usually operated in constant current mode. The mode is automatically switched to the limiting quantity (either voltage or current). Voltage and current can be set by issuing the commands:

```
# Set current to 10 A and voltage to 42 Volt
ptn.set_current(10)
ptn.set_voltage(42)
```

Because the current should not be changed too quickly (the magnetic field would collapse and produce a high voltage spike in reversed polarity), a ramp function was implemented to change the current with a given slope. The same command as before but with slowly rising current.

```
# Set voltage to 42 Volt and current initially to 0 A
ptn.set_voltage(42)
ptn.set_current(0)

# now rise the current with a defined slope of 0.5 A/s
ptn.ramp_current(10, slope=0.5)
```

### Magnetometer

The magnetometer (MagnetPhysik model: FH54) is a hand held device that can be connected to the computer by a serial RS232 interface. The Python library for this device differs in that aspect, that it uses the Python serial implementation and does not use the PyVISA interface. The source code of the library can be found in appendix A.2.8.

To measure the magnetic field, the following commands can be used:

```
# create an instance of the magnetometer and connect to it
magnetometer = FH54()
magnetometer.connect('COM1')

# set the unit and the range
magnetometer.set_unit_Tesla()
magnetometer.set_range(FH54.RANGE_3T)
```

```
# read the value from the magnetometer
magnetic_field = magnetometer.read()
```

The magnetometer can measure magnetic fields up to 3 T and is equipped with a 200 mm long transversal hall probe. The device can either be used battery powered or connected to mains with the included power adapter.

### Function generator

The function generator used in this setup was a Philips PM 5193 capable of generating signals from 0.1 mHz up to 50 MHz. The function generator can be controlled by GPIB interface. The Python library to control the unit can be found in appendix A.2.9.

To set a sine output wave with a frequency of 11 Hz and an amplitude of 1 $V_{rms}$ the following code can be used.

```
# create an instance of the function generator and connect to it
function_generator = PM5193()
function_generator.connect("GPIB0::20::INSTR")

# reset the function generator to the default configuration
function_generator.reset()

# set the waveform parameters
function_generator.set_waveform_sine()
function_generator.set_frequency(11)
function_generator.set_voltage_rms(1)

# enable the output of the device
function_generator.enable_ac()
```

In order to apply also a DC offset to the output the following command can be used:

```
# apply a dc offset of 2.5 V to the output
function_generator.set_dc_offset(2.5)
```

### Keithley 199 Multimeter

The Keithley 199 multimeter can be connected to the computer by GPIB interface. The device was mainly used to measure current or voltages. The basic functionality has been implemented in the Python library that can be found in appendix A.2.10.

To measure DC voltage the following program can be used:

```
from libs.Keithley199 import Keithley199

# connect to the Keithley 199 System DMM
dmm = Keithley199()
dmm.connect("GPIB0::6::INSTR")

# Reset the multimeter to the default configuration
dmm.reset()

# Set function and range
dmm.set_function_dc_volts()
dmm.set_range(Keithley199.RANGE_AUTO)

# Take a measurement
voltage = dmm.measure()
```

### Keithley SourceMeter 2600 series

The Keithley SourceMeter is a source measure unit that can either source or sink current or voltage. Such devices are also called four quadrant devices and are often used to characterize semiconductors. The Keithley SourceMeter has a build in processor and is able to run scripts directly form the device itself.

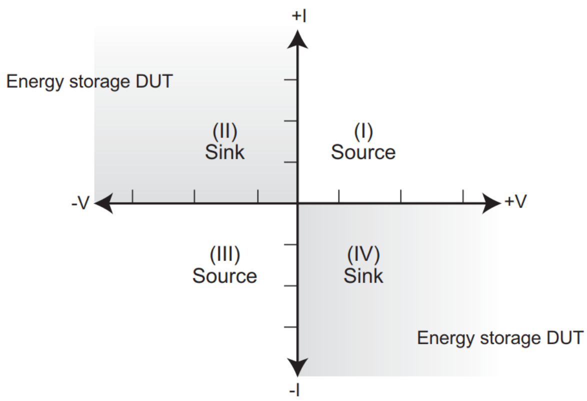



Figure 4.17.: Representation of the four quadrants a source measure unit can operate. The segment (I) represents the quadrant that common power supplies work in [22].

For measurements that are not time critical the functionality can also be used by issuing commands from the computer and transferring the measurement results back to the computer. The Python library that can be found in the appendix A.2.11 implements the basic functionality of the source measure unit.

To perform a simple current and voltage measurement the following script can be used:

```
from KeithleySMU import SMU26xx

# initialize the SMU and connect to it
smu = SMU26xx("TCPIP0::129.27.158.xx::inst0::INSTR")

# get one channel of the SMU (we only need one for this measurement)
smua = smu.get_channel(smu.CHANNEL_A)

# reset to default settings
smua.reset()

# setup the operation mode and what will be shown at the display
smua.set_mode_current_source()
smua.display_voltage()

# define the initial parameters for the channel
smua.set_voltage_range(20)
smua.set_voltage_limit(20)

# set the measurement current to 1 mA
smua.set_current_range(0.1)
smua.set_current_limit(0.1)
smua.set_current(1E-3)

# enable the output
smua.enable_output()

# measure current and voltage simultaneously
[current, voltage] = source_smu.measure_current_and_voltage()
```

### LakeShore 336 Temperature Controller

The LakeShore 336 is used to control the temperature of the cryostat and read out the temperature sensors that are inside. The temperature controller is a PID controller with a 50 W heating element and has multiple sensor inputs. Details about the temperature sensors can be found in chapter 4.2.2.

For temperature control it is advised to use the thermocouple (Sensor D) as PID reference. This sensor is the only build in sensor that covers the whole temperature range from 8 K to 800 K and has little error caused by magnetic fields.

To read the temperature values form the controller, the following commands can be used:

Table 4.1.: Heating power ranges of the LakeShore 336 temperature controller

| range | maximum heating power |
|---|---:|
| off | heater disabled |
| low | 0.5 W |
| medium | 5.0 W |
| high | 50.0 W |

```
from source.libs.LakeShore import Model336

# connect to LakeShore Model 336 temperature controller
temperature_controller = Model336()
temperature_controller.connect("TCPIP0::129.27.158.xx::7777::SOCKET")

# query all temperature sensors simultaneously
[sensor_a, sensor_b, sensor_c, sensor_d] = temperature_controller.
    query_temperatures()

# each sensor can also be queried separately
sensor_a = read_temperature_sensor_A()
sensor_b = read_temperature_sensor_B()
sensor_c = read_temperature_sensor_C()
sensor_d = read_temperature_sensor_D()
```

The output power of the heater can either be set manually (open loop with no feedback) or be controlled by the build in PID controller. The manual operation has the benefit that the heating power does not change and interference with sensitive measurements is less likely. The PID control enables the user to hold a specified temperature or run a specified temperature profile.

If operated above room temperature, the compressor needs to stay turned on to prevent damage to the cold stages of the cold head.

The temperature controller offers different ranges (as listed in table 4.1) for the heating power that limits the maximum output power to the heating element. The range limits the output power regardless if manual output is selected of if the PID control is used.

The manual output specifies the amount of heating in percent in respect to the currently selected range. For example a manual output of 25 % in the range *medium* would result in an output power of 1.25 W as shown in the example code below:

```
# set the output power to 1.25 W
temperature_controller.set_heater_range_medium()
temperature_controller.set_heater1_manual_output(25)
```

The range command is also used to turn the heater off regardless of other setting like set point or manual heating power.

```
# turn the heater off
temperature_controller.set_heater_range_off()
```

To specify a temperature and let the integrated PID controller control the heating element, the following commands can be used. The power to the heating element is now controlled by the PID and is limited by the range. The *.set_heater_range_...* command is also used to turn the heating on.

```
# set the target temperature to 120 K
temperature_controller.disable_setpoint_ramp()
temperature_controller.set_setpoint(120)

# turn on the heater
temperature_controller.set_heater_range_high()
```

In most of the cases a controlled temperature ramp is beneficial. The controller offers a possibility to control the slope of the set point temperature. It is important to know that - if the set point ramp is enabled - all value changes of the set point will be executed with this ramp.

In figure 4.18 the correct and incorrect use of the setpoint feature is described. For this example the assumptions are, that the last user left the system with a setpoint at room temperature (a) and that the current temperature is $T_0 = 8\,K$ (1). For this example we further assume, that a controlled heat up with $\Delta T = 10\,K/min$ to the target temperature $100\,K$ should be achieved.

The wrong usage is, that at $t = 0$ the setpoint ramp is enabled, the target temperature is set and the heater is enabled. The controller immediately starts to constantly change the setpoint temperature with the slope of $\Delta T$ (b) starting from the last value (a) until the target temperature (c) is reached. The temperature PID controller always tries to reach the setpoint temperature. Because the initial setpoint temperature (a) is much higher then the current temperature (1), the temperature controller will do a fast heat up of the system (2). At a certain point the measured temperature is higher then the setpoint

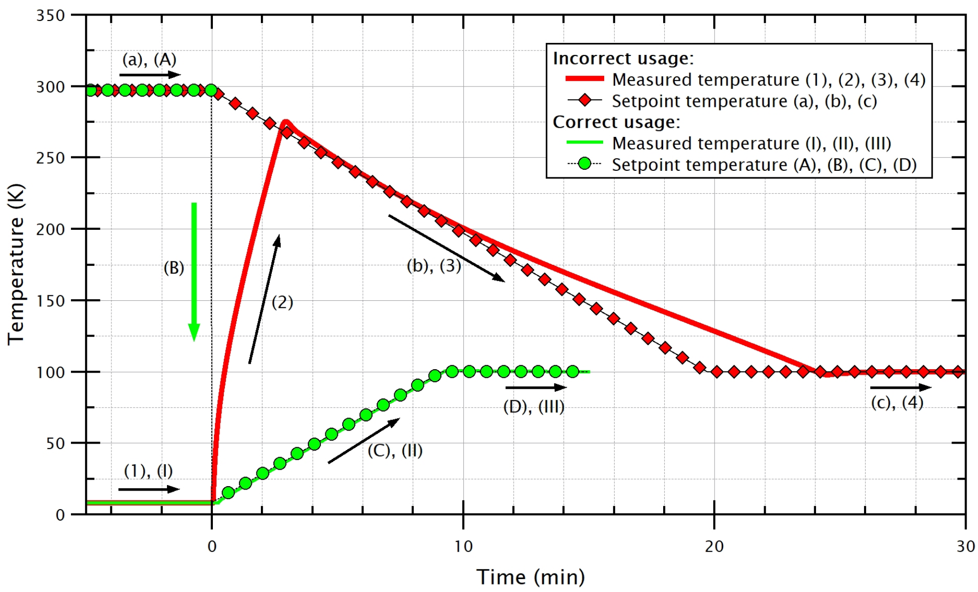



Figure 4.18.: Illustration of the correct and incorrect usage of the setpoint feature. If it is omitted to set the setpoint prior to enabling the ramp (B), the change of the setpoint temperature will start from the last used value (a).

temperature and the heater will shut off in order to match the measured temperature and the setpoint temperature. If the cooling power is higher then the requested slope, the system will, from this point on, follow the changing setpoint until the target temperature (4) is reached. In the case of the example shown in figure 4.18, the cooling power was insufficient to follow the setpoint (b).

The correct way is, to set the the setpoint (A) at $t = 0$ to the current temperature **with disabled ramp** (B). This way the setpoint can instantly change to the current temperature and changes of the setpoint temperature (C) will start from the current temperature (I) until the target temperature (D) is reached. The measured temperature now behaves like expected and rises with the slope $\Delta T$ (II) from the initial value (I) to the target temperature (III).

Correct use of the command if a controlled slope from the current temperature should be achieved:

```
# set the set point to the current temperature with disabled ramp
# this will result in a immediate change in the set_point value
temperature_controller.disable_setpoint_ramp()
current_temperature = temperature_controller.read_temperature_sensor_D()
temperature_controller.set_setpoint(current_temperature)

# set the slope of the heat up to 10 K/s
temperature_controller.set_setpoint_ramp(10)
```

```
# set the target temperature to 100 K
temperature_controller.set_setpoint(100)

# turn the heater on
temperature_controller.set_heater_range_high()
```

## 4.3. Measurement setup

### 4.3.1. Setup at the Graz University of Technology

The simplified measurement setup can be seen in figure 4.19. The setup consists of a AC voltage source that drives a current through the longitudinal axis of the Hall bar type specimen. The voltage source also outputs a DC offset that enables for concurrent AC and DC Hall effect measurements. The lock-in amplifiers are AC coupled so the DC offset will not influence the measurement result of the lock-in amplifiers.

The side arms of the Hall bar are connected to the switch matrix. This enables automated switching of the contacts between resistivity and Hall effect measurements and also offers the possibility to measure across different contact pairs.

In this setup the current is measured with a lock-in amplifier that measures the voltage drop across an external shunt resistor and is also locked to the frequency of the signal generator. The benefit is, that the current measurement and the Hall effect measurement are consistent and also that the current can be reliably measured at low frequencies[8].

The measurement was controlled using a Python script with the measurement technique described in chapter 3.2. A flow diagram of the measurement process is shown in figure 4.20. The Python implementation can be found in appendix A.2.12.

The sample was contacted with 0.1 mm copper wires attached to the ITO with silver loaded conductive epoxy adhesive[9]. The copper wires were mechanically clamped to the electrical contacts of the sample

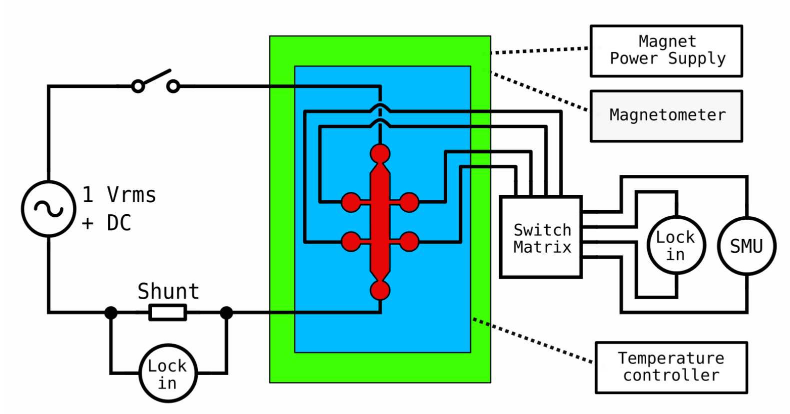



Figure 4.19.: Resistivity and Hall effect measurement setup used for measurements at the new magnetic laboratory showing the connection to a Hall bar type sample. The blue area indicates components of the system that can be cooled and heated. The green area indicates components that reside inside the magnetic field.

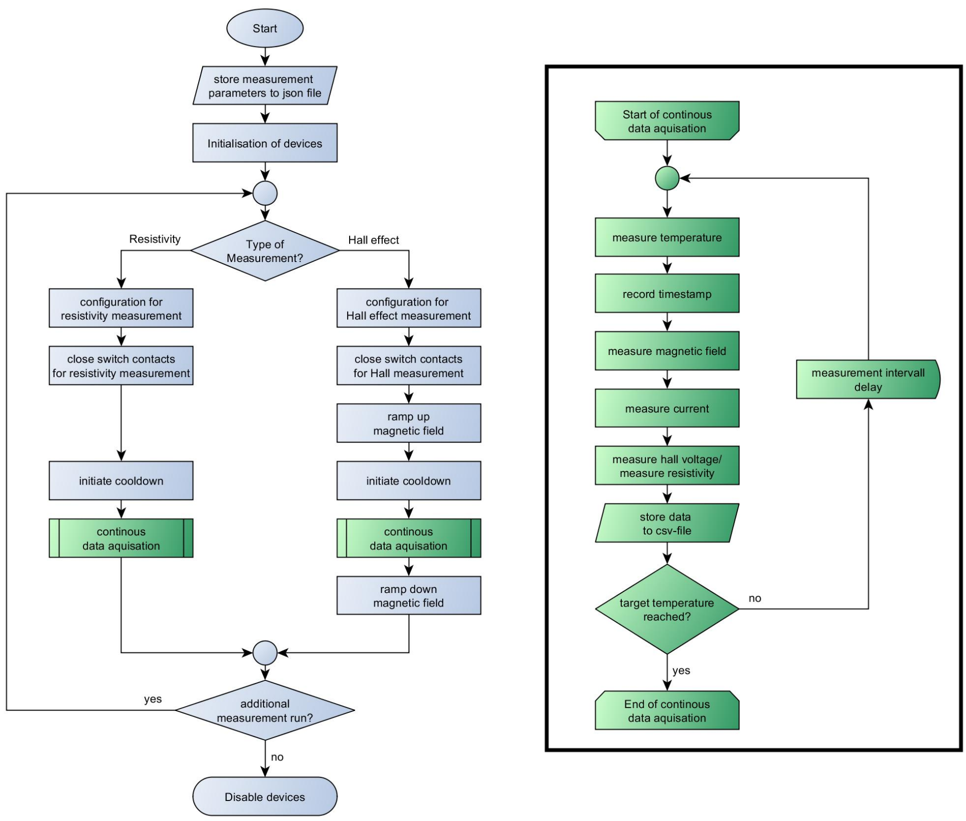



Figure 4.20.: Flow diagram of the Python program to measure Hall coefficient and resistivity during a temperature sweep. The right part of the block diagram shows the process of taking continuous measurements until a specified temperature is reached.

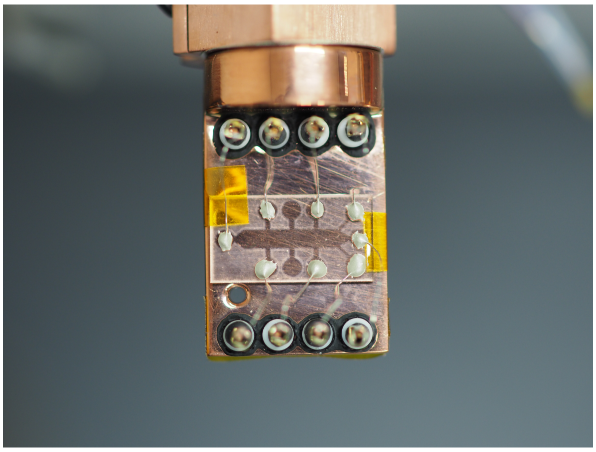


Figure 4.21.: Eight contact Hall bar specimen with contact wires attached with silver loaded conductive epoxy adhesive.

holder. The ITO on glass was fixed with Kapton tape to the sample holder as shown in figure 4.21.

### 4.3.2. Setup at the Ioffe Institute

At the Ioffe Institute (St. Petersburg, Russia) a different approach is used to perform Hall coefficient measurements (a simplified representation is shown in figure 4.22). In comparison to the setup used at the Institute of Solid State Physics (shown in figure 4.19), the Ioffe Institute does not measure the specimen current and the magnetic field but substitutes those measurements with a Hall voltage measurement of a calibrated Hall element.

The index $_{ref}$ represents values associated to the calibrated Hall element (reference). For the calibrated Hall element, the charge carrier density $n_{ref}$ and the thickness $t_{ref}$ are known so equation 2.9 can be rewritten to:

$$U_{ref} = \frac{I_x \, B_z}{e \, n_{ref} \, t_{ref}} \tag{4.1}$$

As the charge carrier density $n_{ref}$ and the thickness $t_{ref}$ of the calibrated Hall element are known, it is possible to substitute those with a constant $C_{ref} = n_{ref} \, t_{ref}$ and rewrite equation 4.1 as:

[8]Multimeters have a minimum and maximum frequency limit for AC measurement capabilities. Outside these limits measurements can be erroneous.

[9]http://at.rs-online.com/web/p/leitende-kleber/1863616/

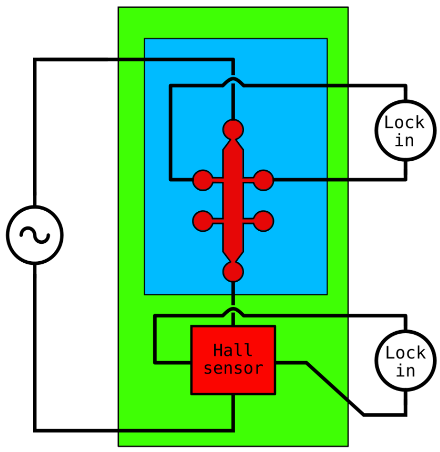



Figure 4.22.: Simplified representation of the setup used at the Ioffe Institute (St. Petersburg, Russia) to perform Hall coefficient measurements. The blue area indicates components of the system that can be cooled and heated. The green area indicates components that reside inside the magnetic field.

$$I_x \, B_z = e \, C_{ref} \, U_{ref} \tag{4.2}$$

The current $I_x$ and the magnetic field $B_z$ are the same for the calibrated Hall element and the device under test. Inserting equation 4.2 into 2.9 results in:

$$\frac{U_H}{U_{ref}} = C_{ref} \, \frac{1}{n \, t} \tag{4.3}$$

As can be seen from equation 4.3, with the ratio of the Hall voltage $U_H$ to the Hall voltage of the calibrated hall element $U_{ref}$, the charge carrier density $n$ of the device under test can be calculated.

# 5. Results

Selected laser structured ITO samples have been measured at the Institute of Solid State Physics, TUGraz and at the Ioffe Institute, St. Petersburg.

The first measurement run at the Ioffe Institute was done from 70 K up to 500 K and back down to room temperature. The resistivity of the ITO shows a temperature dependency and the Hall coefficient stays constant over the whole temperature range as shown in figure 5.1.

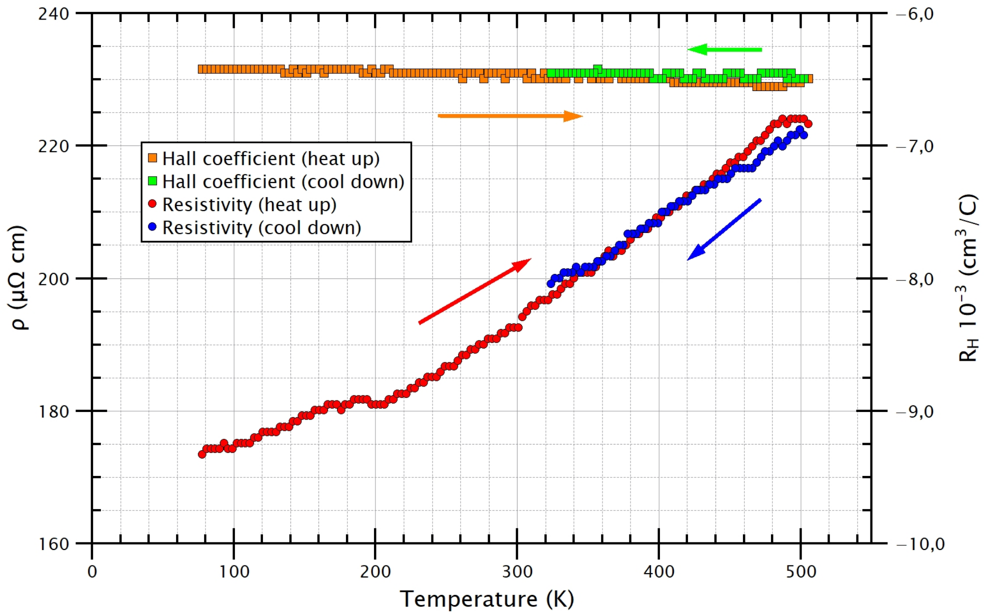



Figure 5.1.: Measurement of the resistivity and Hall coefficient of ITO Hall bar structure. The measurement was performed with alternating magnetic field and alternating current at the Ioffe Institute, Russia.

Resistivity measurements at the Institute of Solid State Physics from 20 K up to room temperature 290 K showed similar results. The measurement was done during a controlled heat up with 10 K/min with the experimental setup as described in figure 4.19. The test voltage for this measurement was 1 $V_{rms}$ and the current was permanently measured and stayed below 3 mA. The measured values (figure 5.2) are in good agreement with the values measured at the Ioffe Institute. The slope of the temperature dependent resistance matches very well between the two Institutes and also the absolute values agree within 5 %. Tuna et al. investigated ITO thin films grown by magnetron sputtering techniques on glass substrate. They produces films with a resistivity of 128 µΩ cm. Their films show a band gap of about 3.64 eV. These values are some of the lowest measured room temperature resistivities reported for both RF and dc sputtered films [23]. Similar results of 145-148 µΩ cm have been achieved by Stowell et al. by RF-superimposed pulsed DC sputtering [24]. The difference in the resistivity values shows, that properties of ITO can be tuned by different manufacturing processes.

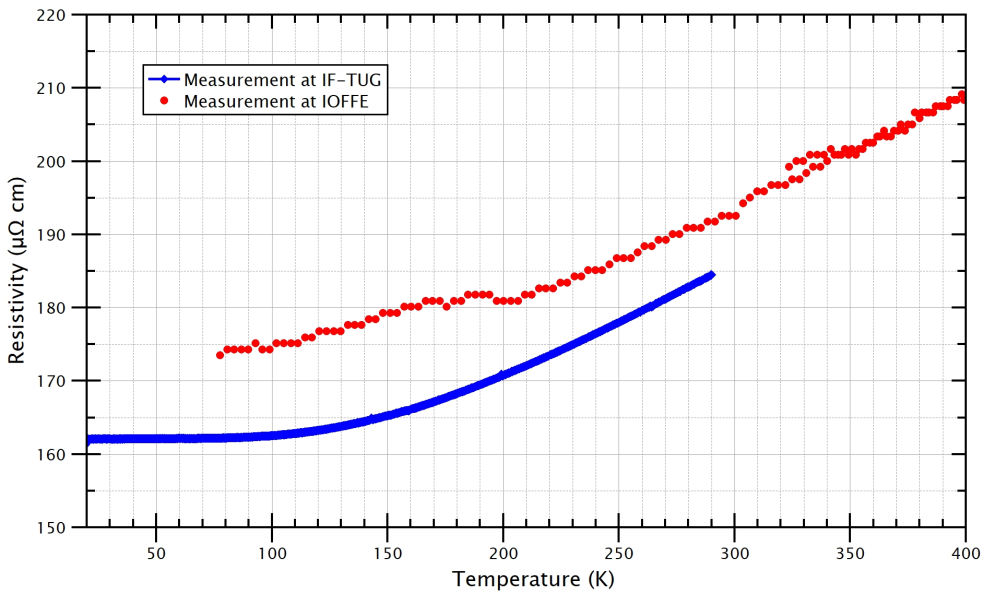



Figure 5.2.: Resistivity measurement of ITO Hall bar structure comparison between results from the Ioffe Institute (red) and the Institute of Solid State Physics (blue).

In another measurement run at the Ioffe Institute form 300 K to 620 K the ITO showed a drastic change in resistivity and Hall coefficient above 550 K. The electrical properties of ITO changed non reversible as was confirmed through cooldown back to room temperature. Nishimoto et al. investigated effects of thermal annealing in relation to the electrical properties of ITO. They showed, that with higher temperatures resistivity increased and charge carrier density decreased [25]. Those findings agree well with the observed changes in resistivity and Hall coefficient found at the Ioffe Institute.

Hall effect measurement done at the Ioffe Institute shows a Hall

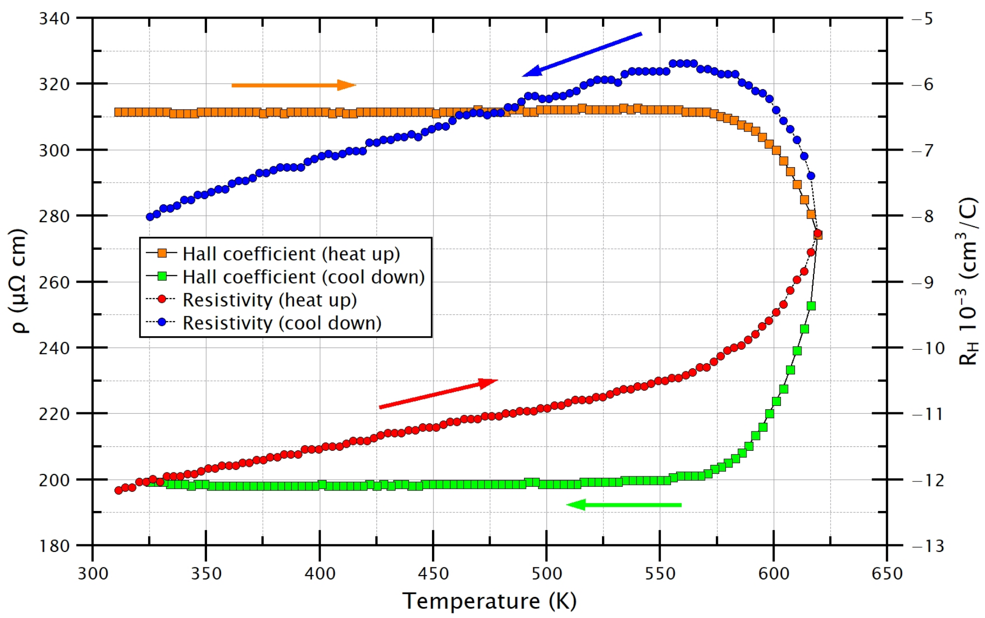



Figure 5.3.: Measurement of the resistivity and Hall coefficient of ITO Hall bar structure from 300 K to 620 K. Above 550 K a non reversible change of the electrical properties of ITO occurred.

constant that is fairly constant over temperature. Measurements at the Institute of Solid State Physics (figure 5.5) show an unexpected temperature dependency. The reason for this behavior is most likely a contact problem at the interface to the sample. As shown in figure 4.2(d) the setup uses a resistor to adjust for geometrical errors. If however one of the contacts the resistor is attached to, changes resistance with temperature differently than the other, this has immediate effect on the current path and therefore on the Hall offset voltage.

Multiple measurement runs were done but contact resistance changed not reproducible. After several temperature runs, the contact to the sample was completely lost. Upon investigation of the sample, hairline fractures could be clearly seen as shown in figure 5.4. A different Hall bar structure was contacted with smaller copper wires (0.1 mm) to reduce physical stress to the contact points by thermal expansion and contraction. The contacting was again done with silver loaded epoxy adhesive that has been cured for 45 min at 80 °C. Also this sample showed a shift in the contact resistance and contact was even lost during the first measurement run. Van Beveren et al. found as a byproduct of their ITO investigation, that wire bonding is possible on ITO thin films [26]. The use of wire bonding should be evaluated to substitute for the silver loaded epoxy adhesive.

Figure 5.4.: Hairline fracture in the silver loaded epoxy adhesive at the contacts to the ITO Hall bar type structure.

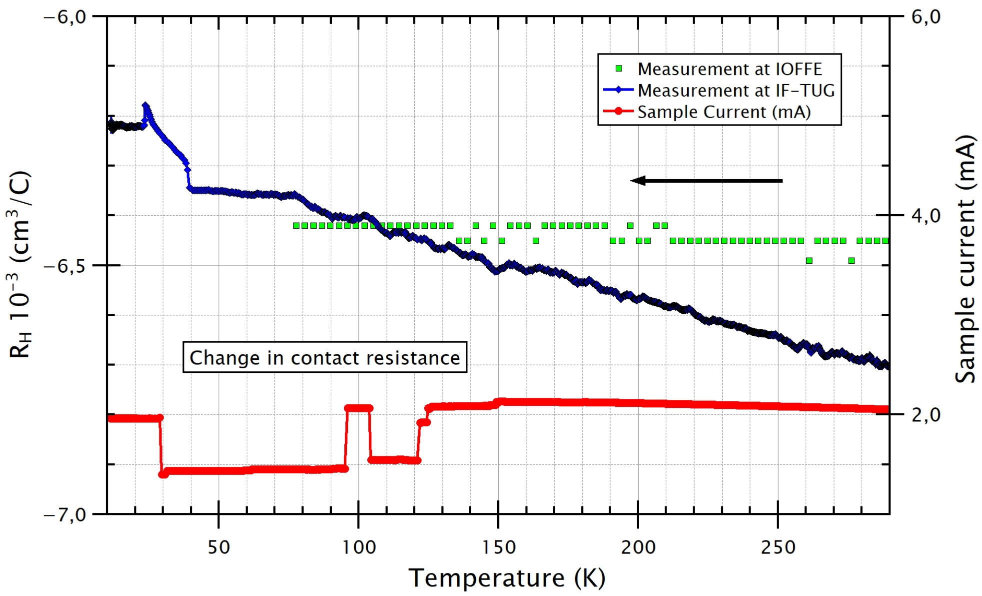



Figure 5.5.: Hall coefficient measurement of ITO Hall bar structure between results from the Ioffe Institute, Russia and the University of Technology, Graz.

Table 5.1.: Hall effect and resistivity measurement results for ITO samples at room temperature. Comparison between results from the TUGraz and the Ioffe Institute.

| | $R_H$ ($cm^3/C$) | $\rho$ ($\mu\Omega$ cm) | n ($cm^{-3}$) | $\mu$ ($cm^2/(V s)$) |
|---|---|---|---|---|
| TUGraz | -6.69 | 184 | $9.33\ 10^{20}$ | 36.35 |
| Ioffe | -6.45 | 193 | $9.68\ 10^{20}$ | 33.52 |
| $\lvert\Delta\rvert$ | 3.7 % | 4.7 % | 3.6 % | 8.4 % |

For room temperature the measurement of the Hall coefficient, the resistivity, the charge carrier density and the charge carrier mobility are in good agreement with the results from the Ioffe Institute as shown in table 5.1. The negative Hall coefficient indicates a n-type semiconductor which is in agreement with known material parameters of ITO [27].

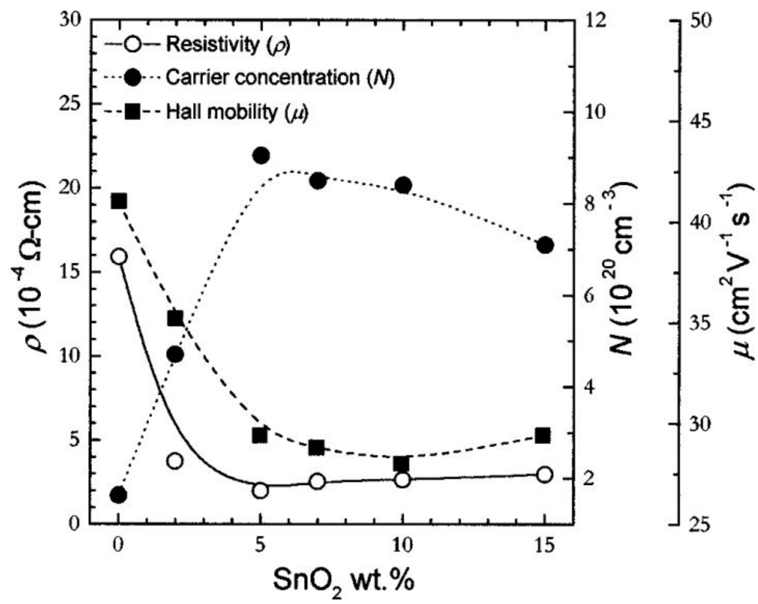



Figure 5.6.: Dependence of resistivity, carrier density, and Hall mobility on SnO2 content for the deposited ITO films. The substrate deposition temperature was kept at 250 °C and the oxygen pressure was 10 mTorr during deposition [15].

The measured values also are in the same order of magnitude that Kim et al. [15] found upon investigation of different deposition temperatures of ITO as shown in figure 5.6. Van Beveren et al. growed ITO thin films on silica substrates. For a 12.5 nm PVD annealed thin film they measured a room temperature resistivity of 815 μΩ cm. The electron density equals $3.6\ 10^{20}\ \mathrm{cm}^{-3}$ and results in a mobility of 21.3 cm²/(V s) [26].

The deviation of electrical properties of ITO among different research group shows, that these values are highly dependent on the exact process of manufacturing. Pern found that ITO even shows degradation if exposed to damp humid air (80 °C, 85 % RH) for several hours [28].

# 6. Conclusions

The realization of a laboratory setup for Hall effect measurements was successfully performed. The existing magnet has been characterized and is capable of providing a constant magnetic field up to 1 T with a field homogeneity better than 99.9 % across the whole sample-area. The closed cycle cryostat can reach a base temperature of 8 K in less then 70 min and the self heat up duration, from base temperature to room temperature, is longer than 12 h at a thermal insulation vacuum of $10^{-6}$ mbar. The closed cycle cryostat is also equipped with a heating interface enabling for temperature dependent measurements up to 800 K.

The electrical measurement setup provides the possibility to perform simultaneous measurement of electrical resistivity and Hall coefficient and uses a combined AC / DC method to enhance accuracy. The usage of the lock-in measurement technique enables reliable measurements of Hall voltages down to the nV range. Thin film specimens with dimensions up to 20 x 15 mm that have an electrical resistance between 0.1 Ω and 10 MΩ can be measured with this setup. Preparations for measurements on high impedance specimens with an electrical resistance larger than 10 MΩ have been made and can be easily implemented. For all measurement devices, a Python library was developed to provide easy access to device functionalities. The Python library uses the National Instruments Virtual Instrument Software Architecture (NI-VISA) to provide connection independent and operating system independent access to measurement device functions. Multiple Python programs have been written to perform resistivity and Hall effect measurements and fully automate the testing process.

Different types of Hall standard specimen geometries were produced by laser structuring Indium Tin Oxide (ITO) material. The laser structuring process provides high geometrical accuracy of ±2 µm. The ITO samples were measured by spectroscopic ellipsometry and the evaluated thickness is $(78 \pm 2)$ nm. The specimens have been investigated in their electrical properties at the Russian Academy of Sciences, Ioffe Physical Technical Institute in the temperature range between 70 K and 620 K. For room temperature, the ITO showed an electrical resistivity of $\rho$ = 193 μΩ cm, a charge carrier density of $n$ = 9.68 $10^{20}$ $cm^{-3}$ and a charge carrier mobility of $\mu$ = 33.52 $cm^2/(V\ s)$.

Temperature dependent resistivity measurements between 8 K and 300 K of the ITO thin film samples reveal an increasing electrical resistivity with temperature, correlating to the behavior expected from a classical conductor. Hall effect measurements have been made at room temperature with a test current of $I_x$ = 2 mA at a magnetic field of B = 1 T resulting in a Hall voltage of $U_H$ = -170 μV that indicates a n-type material with electrons acting as charge carriers. The ITO standard samples show an electrical resistivity of $\rho$ = 184 μΩ cm, a charge carrier density of $n$ = 9.33 $10^{20}$ $cm^{-3}$ and a charge carrier mobility of $\mu$ = 36.35 $cm^2/(V\ s)$.

# Appendix

# Appendix A.

# Appendix

## A.1. Device specifications

### A.1.1. Bruker Electromagnet

**Manufacturer:** Bruker Austria GmbH
**Type:** B-E15 B8
**Serial:** 158 1118
**TUGraz Inventory number:** 0071410
**Maximum cooling water pressure:** 3 bar
**Maximum continuous current:** 30 A per coil
**Resistance:** 1.5 Ω per coil
**Wiring:** Coils are in series
**Remarks:** The magnet is equipped with reverse diodes to protect from voltage spikes upon collapsing magnetic fields.

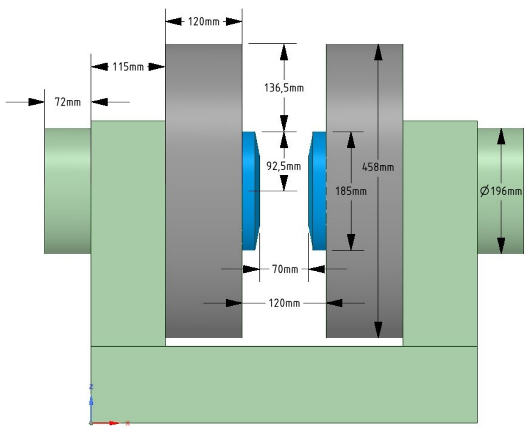



Figure A.1.: Dimensions of the Bruker B-E15 electromagnet in side view without the supporting frame.

## A.1.2. Heinzinger Power Supply

Technical specifications are taken form the device manual [29].

> **Manufacturer:** Heinzinger electronic GmbH
> **Type:** PTN 125-40
> **Part number:** 00.220.230.1-02
> **Serial:** 3352 09878
> **TUGraz inventory number:** 0113868

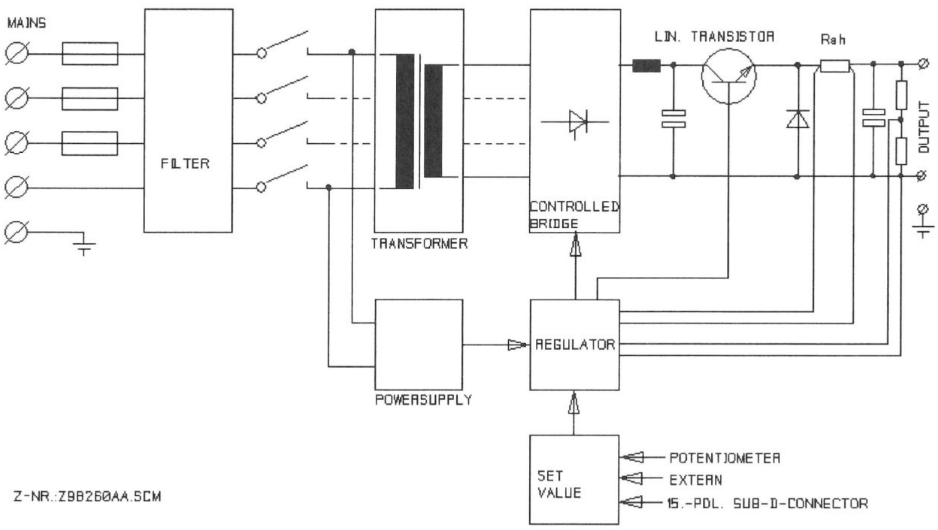



Figure A.2.: Principle schematic of operation of the Heinzinger PTN 125-40 power supply.

# 4. Technical Data

Technical data of the power pack - like maximal output voltage ($U_{NENN}$) and maximal output current ($I_{NENN}$) - can be seen in the type designation. The first number after the series name stands for the nominal voltage in volt, the second number for the nominal current in milliampere
For example: PTN 125 - 5 means $U_N$ = 125 V, $I_N$ = 5 mA. Mains supply voltage and fuse rate can be seen on the device type plate at the back of the power pack.

| | |
|---|---|
| **General:** | |
| **Mains connection** | |
| **1-phase-devices** | 230V ± 10%, 47 .. 63 Hz |
| **2-phase-devices** | 400V three-phase current ±10%, 47 .. 63 Hz |
| **Ambient temperature:** | 0°C .. +40°C |
| **Potential isolation output** | according to VDE 0160 |
| **Discharge time (open output)** | depend on type <30s for voltages smaller then 50V |

**Voltage Stabilization:**

| | |
|---|---|
| **Setting range:** | from approx. 0,1% to 100% $U_{nom}$ (no-load operation stable operation > 1% to 5%) |
| **Accuracy of setting range:** | ±0,02% $U_{nom}$ |
| **Reproducibility:** | ±0,05% $U_{nom}$ |
| **Reproducibility at ±10% mains variation:** | <±0,001% $U_{nom}$ |
| **Between no-load and full-load operation:** | <±0,01% $U_{nom}$ ±200µV |
| **Control time (no-load to full-load operation):** | <5ms to 0,1% $U_{nom}$ deviation (type-dependent) |
| **Stability (over period of more then 8 hrs., under constant operating conditions)** | |
| **PTN Series** | ≤0,01% $U_{nom}$ |
| **PTN*hp*-Series** | ≤0,001% $U_{nom}$ |
| **Temperature coefficient** | |
| **PTN Series** | ≤0,01% $U_{nom}$ /K |
| **PTN*hp*-Series** | ≤0,001% $U_{nom}$ /K |
| **Residual ripple:** | |
| **PTN Series** | ≤0,01% pp ±1mV $U_{nom}$ |
| **PTN*hp*-Series** | ≤0,001% pp ±500µV $U_{nom}$ |

**Current Stabilization:**

| | |
|---|---|
| **Setting range:** | from approx. 0,1% to 100% $I_{nom}$ |
| **Accuracy of setting range:** | ±0,02% $I_{nom}$ |
| **Reproducibility** | ±0,05% $I_{nom}$ |
| **Reproducibility at ±10% mains variation:** | <±0,003% $I_{nom}$ ±200µA |
| **Reproducibility at ±10% load variation:** | <±0,01% $I_{nom}$ ±100µA |
| **Control time (±10%$\Delta R_L$)** | <5ms to 0,1% Inom deviation (type-dependent) |
| **Stability (over period of more then 8 hrs, under constant operating conditions)** | |
| **PTN Series** | ≤0,02% $I_{nom}$ |
| **PTN*hp*-Series** | ≤0,002% $I_{nom}$ |
| **Temperature coefficient** | |
| **PTN Series** | ≤0,02% $I_{nom}$ /K |
| **PTN*hp*-Series** | ≤0,002% $I_{nom}$ /K |
| **Residual ripple:** | |
| **PTN Series** | ≤0,05% pp ±1mA $I_{nom}$ |
| **PTN*hp*-Series** | ≤0,005% pp ±1mA $I_{nom}$ |

Above technical specifications can differ from customized equipment.

### A.1.3. Advanced Research Systems Cryostat

The cryogenic-system consists of a cold head, a helium compressor and a temperature controller. Technical information presented here was taken from documents that came with the shipping or from the provided user manuals [30], [31], [32].

**Cold-head:**

**Manufacturer:** Advanced Research Systems, Inc
**Type:** CS202AE-DMX-3-1AL
**Serial:** 16-E1868
**TUGraz inventory number:** to be assigned

**Helium compressor:**

**Manufacturer:** Advanced Research Systems, Inc
**Type:** ARS-4HW
**Serial:** 16-HC1438ST
**TUGraz inventory number:** to be assigned

**Temperature controller:**

**Manufacturer:** Lake Shore Cryotronics, Inc.
**Type:** Model 336
**Serial:** LSA17UL
**TUGraz inventory number:** 540289/2017/0001

# Non-Optical Cryostat - Economy

The **CS202*E-DMX-3-1AL** offers a wide range of flexibility at a low cost, making it an excellent choice for most sample and device testing. This system is well suited for optical, electrical, and magnetic sample testing.

## Applications

- Resistivity/Hall Probe Experiments
- Thermal, Electrical and Magnetic Susceptibility
- Heat Capacitance
- Seebeck Effect
- DLTS

## Features

- Cryogen Free, Low Power
- Low cost aluminum construction
- Can operate in any orientation
- Fully customizable

## Typical Configuration

- Cold head (DE-202AE)
- Compressor (ARS-2HW)
- 2 Helium Hoses
- Aluminum vacuum shroud for electrical experiments (DMX-3)
- Aluminum radiation shield
- Instrumentation for temperature measurement and control:
  - 10 pin hermetic feed through
  - 36 ohm thermofoil heater
  - Silicon diode sensor curve matched to (±0.5K) for control
  - Calibrated silicon diode sensor (±12 mk) with 4 in. free length for accurate sample measurement.
- Wiring for electrical experiments:
  - 10 pin hermetic feed through
  - 4 copper wires
- Sample holder for electrical experiments
- Temperature Controller

## Options and Upgrades

- 4K Coldhead (0.1W @ 4.2K)
- 5.5K Coldhead (1W @ 10K)
- 450K High Temperature Interface
- 800K High Temperature Interface
- Turbo upgrade for faster cooldown times
- Custom temperature sensor configuration (please contact our sales staff
- Custom wiring configurations (please contact our sales staff)
- Window material upgrades (custom materials available)
- Sample holder upgrades (custom sample holders available)

The above picture shows a cryocooler with a vacuum shroud, radiation shield, and sample holder installed.

The above picture shows a complete system (minus the vacuum pump and temperature controller)

DS-CS202*E-DMX-3-1-R1

www.arscryo.com

# Non-Optical Cryostat - Economy

**Cooling Technology-**

| | |
|---|---|
| DE-202 | Closed Cycle Cryocooler |
| Refrigeration Type | Pneumatically Driven GM Cycle |
| Liquid Cryogen Usage | None, Cryogen Free |

**Temperature*-**

| | |
|---|---|
| DE-202AE | < 10K - 350K |
| DE-202SE | < 4.2K - 350K |
| DE-202PE | < 5.5K - 350K |
| With 800K Interface | (Base Temp + 2K) - 700K |
| With 450K Interface | (Base Temp + 2K) - 450K |
| Stability | 0.1K |
| *Based on bare cold head with a closed radiation shield, and no additional sources of experimental or parasitic heat load | |

**Sample Space -**

| | |
|---|---|
| Diameter | 36 mm (1.43 in.)27mm(1.06in) |
| Height | 39 mm (1.53 in.) |
| Sample Holder Attachment | 1/4 - 28 screw |
| Sample Holder | www.arscryo.com/Products/SampleHolders.html |

**Optical Access-**

| | |
|---|---|
| Window Ports | N/A |
| Diameter | N/A |
| Clear View | N/A |
| #/F | N/A |
| Window Material | N/A |

**Temperature Instrumentation and Control - (Standard) -**

| | |
|---|---|
| Heater | 36 ohm Thermofoil Heater anchored to the coldtip |
| Control Sensor | Curve Matched Silicon Diode installed on the coldtip |
| Sample Sensor | Calibrated Silicon Diode with free length wires |
| Contact ARS for other options | |

**Instrumentation Access-**

| | |
|---|---|
| Instrumentation Skirt | Bolt-On, Aluminum |
| Pump out Port | 1 - NW 25 |
| Instrumentation Ports | 2 |
| Instrumentation Wiring | Contact sales staff for options |

**Vacuum Shroud -**

| | |
|---|---|
| Material | Aluminum |
| Length | 338 mm (13.3 in) |
| Diameter | 45 mm (1.75 in) at the sample space |
| | 35mm (1.37 in) FMX-3-1B |

**Radiation Shield -**

| | |
|---|---|
| Material | Aluminum |
| Attachment | Threaded |
| Optical Access | N/A |

**Cryostat Footprint -**

| | |
|---|---|
| Overall Length | 544 mm (21.41 in) |
| Motor Housing Diameter | 114 mm (4.5 in) |
| Rotational Clearance | 200 mm (8 in) with "G" Configuration |

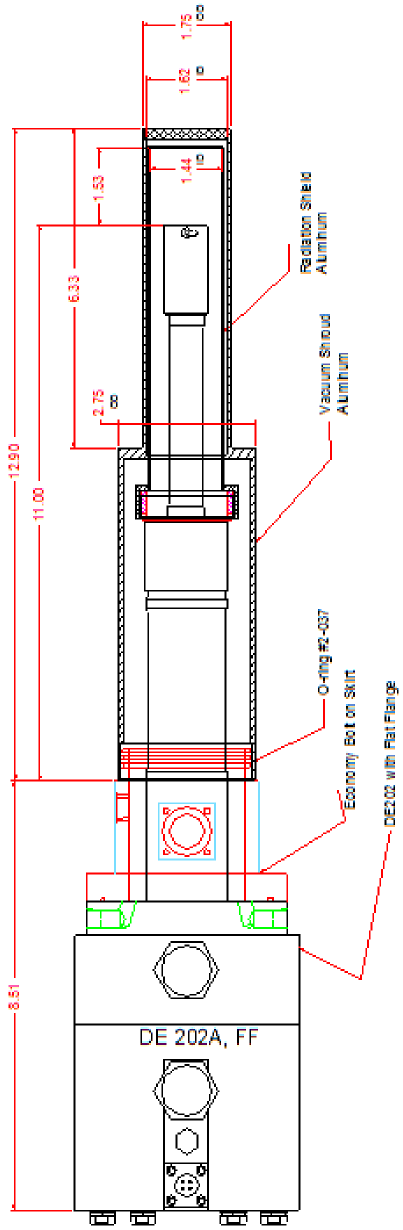
| **Cryocooler Model** | | **DE-202AE** | | **DE-202A(T)E** | | **DE-202PE** | | **DE-202SE** | |
|---|---|---|---|---|---|---|---|---|---|
| | Frequency | 60 Hz | 50 Hz | 60 Hz | 50 Hz | 60 Hz | 50 Hz | 60 Hz | 50 Hz |
| **Base Temperature** | | <9K | <9K | <9K | <9K | <5.5K | <5.5K | <4.2K | <4.2K |
| **Cooling Capacity** | 4.2K | - | - | - | - | - | - | 0.1W | 0.08W |
| | 10K | 0.5W | 0.4W | 0.7W | 0.56W | 1W | 0.8W | 1.2W | 1W |
| | 20K | 2.5W | 2W | 3.7W | 3W | 3.5W | 2.8W | 4W | 3.2W |
| | 77K | 4W | 3.2W | 6W | 4.8W | 3.5W | 2.8W | 4W | 3.2W |
| **Radiation Shield Cooling Capacity** | | 10W | 8W | 15W | 12W | 10W | 8W | 10W | 8W |
| **Cooldown Time** | 20K | 50 min | 60 min | 35 min | 42 min | 60 min | 72 min | 60 min | 72 min |
| | Base Temperature | 70 min | 84 min | 50 min | 60 min | 90 min | 108 min | 90 min | 108 min |
| **Compressor Model** | | ARS-2HW | | ARS-2HW | | ARS-2HW | | ARS-4HW | |
| **Typical Maintenance Cycle** | | 12,000 hours | | 8,000 hours | | 12,000 hours | | 12,000 hours | |

DS-CS202*E-DMX-3-1-R1

www.arscryo.com

# Non-Optical Cryostat - Economy

**DE202*E-DMX-3-1 Outline Drawing**

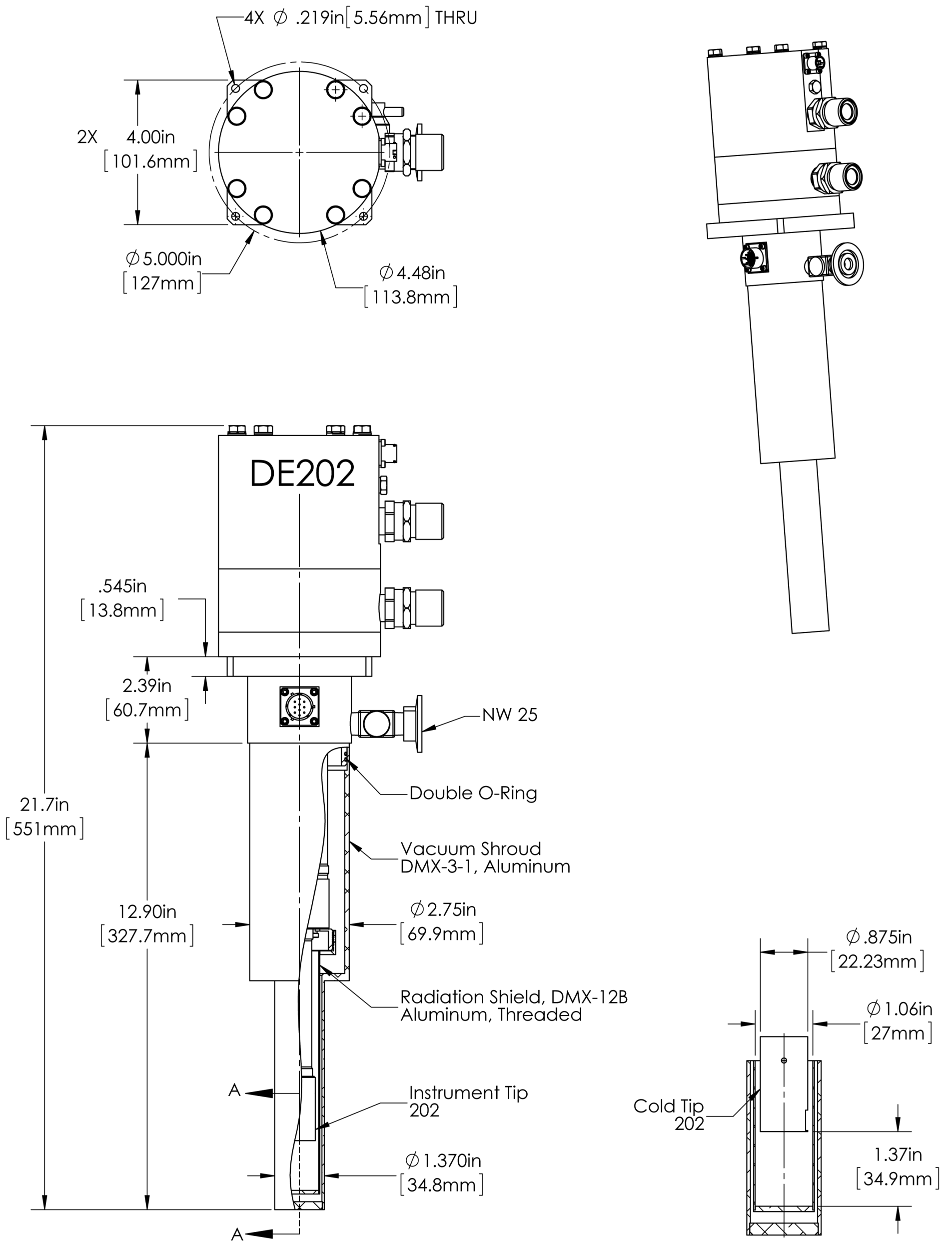



DS-CS202*E-DMX-3-1-R1

www.arscryo.com

# Non-Optical Cryostat - Economy

**DE202*E-DMX-3-1B Outline Drawing**



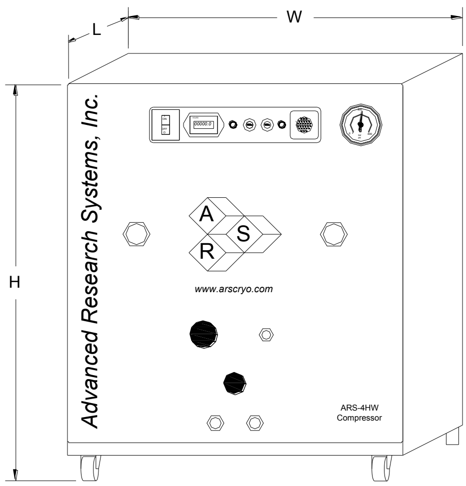

DS-CS202*E-DMX-3-1-R1

www.arscryo.com

# Non-Optical Cryostat - Economy

**Sample Space**

**ARS-2HW/ARS-4HW Compressor**




| Compressor Model | | ARS-2HW | | ARS-4HW | |
|---|---|---|---|---|---|
| | Frequency | 60 Hz | 50 Hz | 60 Hz | 50 Hz |
| **Standard Voltage** | Min | 208 V | 190 V | 208 V | 190 V |
| | Max | 230 V | 210 V | 230 V | 210 V |
| **Transformer Options** | 10% | | 220 V, 230V | | 220 V, 230 V |
| | 15% | | 240 V | | 240 V |
| **Power Usage** | Single Phase | 1.3 kW | 1.2 kW | 3.6 kW | 3.0 kW |
| **Refrigerant Gas** | | 99.999% Helium Gas, Pre-Charged | | 99.999% Helium Gas, Pre-Charged | |
| **Noise Level** | | 60 dBA | | 60 dBA | |
| **Ambient Temperature** | | 12 - 40 C (54—104 F) | | | |
| **Cooling Water** | Consumption | 1.5 L / min (0.4 Gal. / min) | | 2.3 L / min (0.6 Gal. / min) | |
| | Temperature | 10 - 35 C (50—95 F) | | 10 - 35 C (50—95 F) | |
| | Connection | 3/8 in. Swagelok Fitting | | 3/8 in. Swagelok Fitting | |
| **Dimensions:** | L | 483 mm (19 in) | | 483 mm (19 in) | |
| | W | 434 mm (17.1 in) | | 434 mm (17.1 in) | |
| | H | 516 mm (20.3 in) | | 516 mm (20.3 in) | |
| **Weight** | | 62 kg (137 lbs) | | 72 kg (160 lbs) | |
| **Typical Maintenance Cycle** | | 12,000 hours | | 12,000 hours | |
| **Water Recirculation Option** | | CoolPac Compatible | | CoolPac Compatible | |

DS-CS202*E-DMX-3-1-R1 www.arscryo.com

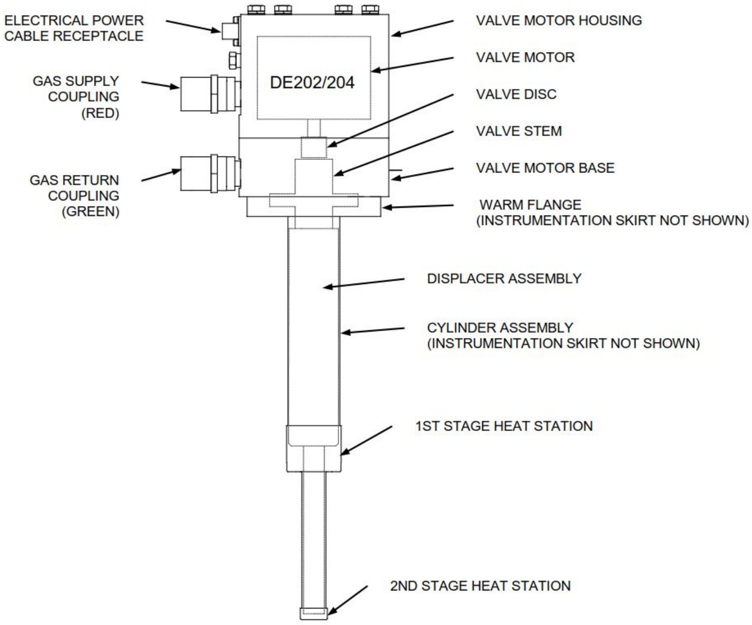



Figure A.3.: Description of the construction and components inside the cold head of the ARS CS202AE-DMX-3-1AL [30].

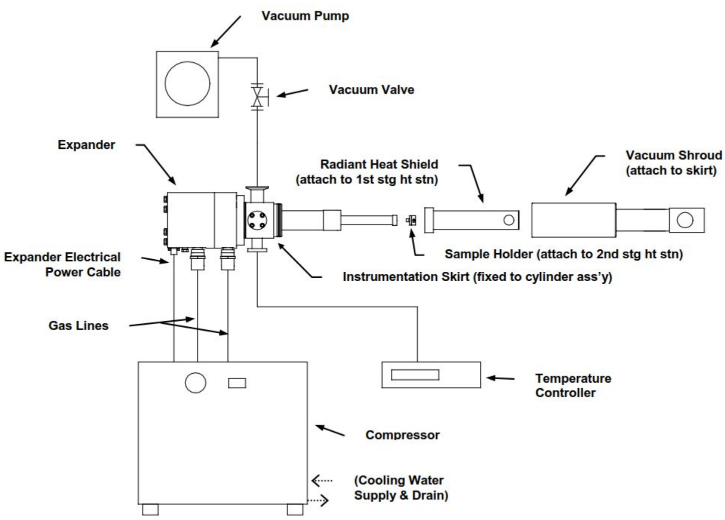



Figure A.4.: Typical setup of the ARS cryostat components interconnections. Showing a simple shroud, radiation shield and sample holder [30].

# Non-Optical Cryostat - Economy

The **CS202*E-DMX-3-1AL** offers a wide range of flexibility at a low cost, making it an excellent choice for most sample and device testing. This system is well suited for optical, electrical, and magnetic sample testing.

## Applications

- Resistivity/Hall Probe Experiments
- Thermal, Electrical and Magnetic Susceptibility
- Heat Capacitance
- Seebeck Effect
- DLTS

## Features

- Cryogen Free, Low Power
- Low cost aluminum construction
- Can operate in any orientation
- Fully customizable

## Typical Configuration

- Cold head (DE-202AE)
- Compressor (ARS-2HW)
- 2 Helium Hoses
- Aluminum vacuum shroud for electrical experiments (DMX-3)
- Aluminum radiation shield
- Instrumentation for temperature measurement and control:
  - 10 pin hermetic feed through
  - 36 ohm thermofoil heater
  - Silicon diode sensor curve matched to (±0.5K) for control
  - Calibrated silicon diode sensor (±12 mk) with 4 in. free length for accurate sample measurement.
- Wiring for electrical experiments:
  - 10 pin hermetic feed through
  - 4 copper wires
- Sample holder for electrical experiments
- Temperature Controller

## Options and Upgrades

- 4K Coldhead (0.1W @ 4.2K)
- 5.5K Coldhead (1W @ 10K)
- 450K High Temperature Interface
- 800K High Temperature Interface
- Turbo upgrade for faster cooldown times
- Custom temperature sensor configuration (please contact our sales staff
- Custom wiring configurations (please contact our sales staff)
- Window material upgrades (custom materials available)
- Sample holder upgrades (custom sample holders available)

The above picture shows a cryocooler with a vacuum shroud, radiation shield, and sample holder installed.

The above picture shows a complete system (minus the vacuum pump and temperature controller)

DS-CS202*E-DMX-3-1-R1

www.arscryo.com

# Non-Optical Cryostat - Economy

**Cooling Technology-**

| | |
|---|---|
| DE-202 | Closed Cycle Cryocooler |
| Refrigeration Type | Pneumatically Driven GM Cycle |
| Liquid Cryogen Usage | None, Cryogen Free |

**Temperature*-**

| | |
|---|---|
| DE-202AE | < 10K - 350K |
| DE-202SE | < 4.2K - 350K |
| DE-202PE | < 5.5K - 350K |
| With 800K Interface | (Base Temp + 2K) - 700K |
| With 450K Interface | (Base Temp + 2K) - 450K |
| Stability | 0.1K |
| *Based on bare cold head with a closed radiation shield, and no additional sources of experimental or parasitic heat load | |

**Sample Space -**

| | |
|---|---|
| Diameter | 36 mm (1.43 in.)27mm(1.06in) |
| Height | 39 mm (1.53 in.) |
| Sample Holder Attachment | 1/4 - 28 screw |
| Sample Holder | www.arscryo.com/Products/SampleHolders.html |

**Optical Access-**

| | |
|---|---|
| Window Ports | N/A |
| Diameter | N/A |
| Clear View | N/A |
| #/F | N/A |
| Window Material | N/A |

**Temperature Instrumentation and Control - (Standard) -**

| | |
|---|---|
| Heater | 36 ohm Thermofoil Heater anchored to the coldtip |
| Control Sensor | Curve Matched Silicon Diode installed on the coldtip |
| Sample Sensor | Calibrated Silicon Diode with free length wires |
| Contact ARS for other options | |

**Instrumentation Access-**

| | |
|---|---|
| Instrumentation Skirt | Bolt-On, Aluminum |
| Pump out Port | 1 - NW 25 |
| Instrumentation Ports | 2 |
| Instrumentation Wiring | Contact sales staff for options |

**Vacuum Shroud -**

| | |
|---|---|
| Material | Aluminum |
| Length | 338 mm (13.3 in) |
| Diameter | 45 mm (1.75 in) at the sample space |
| | 35mm (1.37 in) FMX-3-1B |

**Radiation Shield -**

| | |
|---|---|
| Material | Aluminum |
| Attachment | Threaded |
| Optical Access | N/A |

**Cryostat Footprint -**

| | |
|---|---|
| Overall Length | 544 mm (21.41 in) |
| Motor Housing Diameter | 114 mm (4.5 in) |
| Rotational Clearance | 200 mm (8 in) with "G" Configuration |

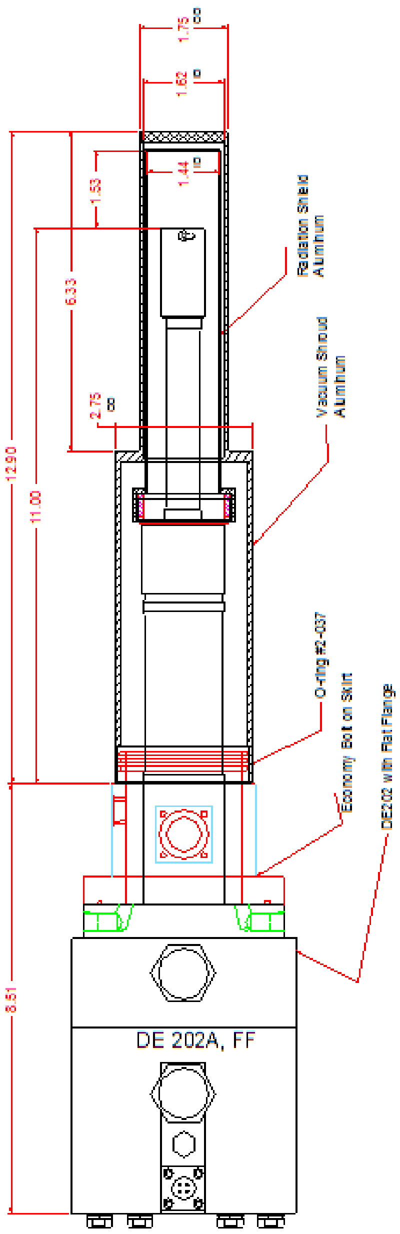
| **Cryocooler Model** | | **DE-202AE** | | **DE-202A(T)E** | | **DE-202PE** | | **DE-202SE** | |
|---|---|---|---|---|---|---|---|---|---|
| | Frequency | 60 Hz | 50 Hz | 60 Hz | 50 Hz | 60 Hz | 50 Hz | 60 Hz | 50 Hz |
| **Base Temperature** | | <9K | <9K | <9K | <9K | <5.5K | <5.5K | <4.2K | <4.2K |
| **Cooling Capacity** | 4.2K | - | - | - | - | - | - | 0.1W | 0.08W |
| | 10K | 0.5W | 0.4W | 0.7W | 0.56W | 1W | 0.8W | 1.2W | 1W |
| | 20K | 2.5W | 2W | 3.7W | 3W | 3.5W | 2.8W | 4W | 3.2W |
| | 77K | 4W | 3.2W | 6W | 4.8W | 3.5W | 2.8W | 4W | 3.2W |
| **Radiation Shield Cooling Capacity** | | 10W | 8W | 15W | 12W | 10W | 8W | 10W | 8W |
| **Cooldown Time** | 20K | 50 min | 60 min | 35 min | 42 min | 60 min | 72 min | 60 min | 72 min |
| | Base Temperature | 70 min | 84 min | 50 min | 60 min | 90 min | 108 min | 90 min | 108 min |
| **Compressor Model** | | ARS-2HW | | ARS-2HW | | ARS-2HW | | ARS-4HW | |
| **Typical Maintenance Cycle** | | 12,000 hours | | 8,000 hours | | 12,000 hours | | 12,000 hours | |

DS-CS202*E-DMX-3-1-R1

**www.arscryo.com**

# Non-Optical Cryostat - Economy

**DE202*E-DMX-3-1 Outline Drawing**

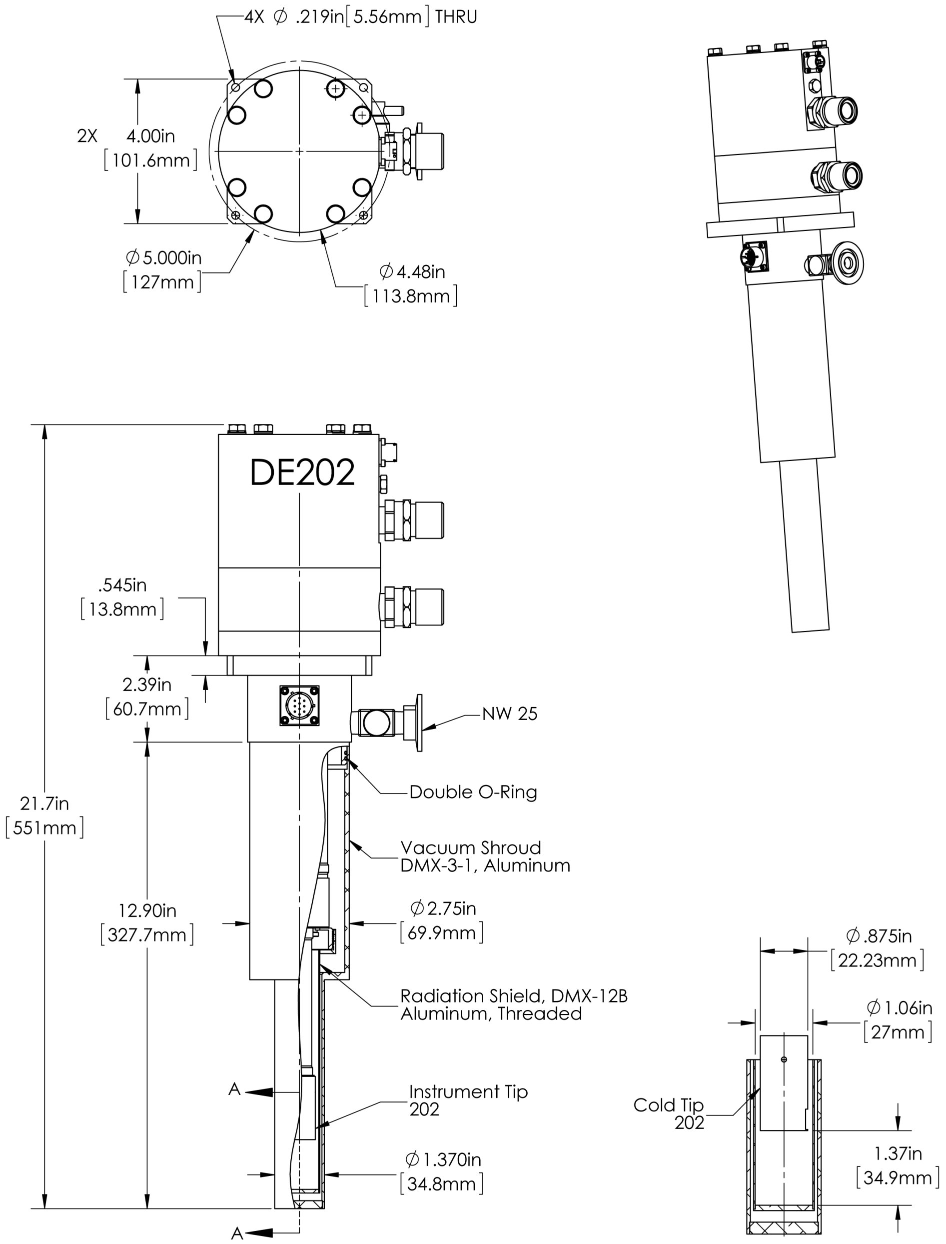



DS-CS202*E-DMX-3-1-R1

www.arscryo.com

# Non-Optical Cryostat - Economy

**DE202*E-DMX-3-1B Outline Drawing**



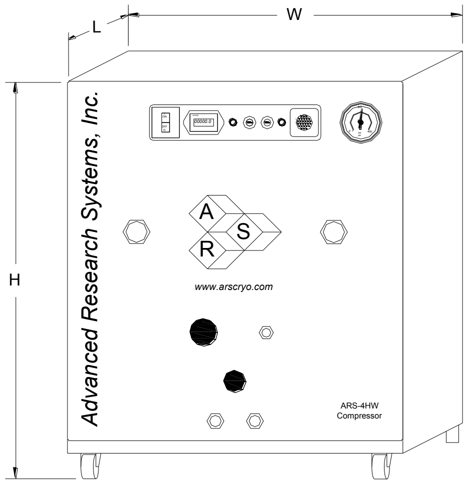

DS-CS202*E-DMX-3-1-R1

*www.arscryo.com*

# Non-Optical Cryostat - Economy

**Sample Space**

**ARS-2HW/ARS-4HW Compressor**




| Compressor Model | | ARS-2HW | | ARS-4HW | |
|---|---|---|---|---|---|
| | Frequency | 60 Hz | 50 Hz | 60 Hz | 50 Hz |
| **Standard Voltage** | Min | 208 V | 190 V | 208 V | 190 V |
| | Max | 230 V | 210 V | 230 V | 210 V |
| **Transformer Options** | 10% | | 220 V, 230V | | 220 V, 230 V |
| | 15% | | 240 V | | 240 V |
| **Power Usage** | Single Phase | 1.3 kW | 1.2 kW | 3.6 kW | 3.0 kW |
| **Refrigerant Gas** | | 99.999% Helium Gas, Pre-Charged | | 99.999% Helium Gas, Pre-Charged | |
| **Noise Level** | | 60 dBA | | 60 dBA | |
| **Ambient Temperature** | | 12 - 40 C (54—104 F) | | | |
| **Cooling Water** | Consumption | 1.5 L / min (0.4 Gal. / min) | | 2.3 L / min (0.6 Gal. / min) | |
| | Temperature | 10 - 35 C (50—95 F) | | 10 - 35 C (50—95 F) | |
| | Connection | 3/8 in. Swagelok Fitting | | 3/8 in. Swagelok Fitting | |
| **Dimensions:** | L | 483 mm (19 in) | | 483 mm (19 in) | |
| | W | 434 mm (17.1 in) | | 434 mm (17.1 in) | |
| | H | 516 mm (20.3 in) | | 516 mm (20.3 in) | |
| **Weight** | | 62 kg (137 lbs) | | 72 kg (160 lbs) | |
| **Typical Maintenance Cycle** | | 12,000 hours | | 12,000 hours | |
| **Water Recirculation Option** | | CoolPac Compatible | | CoolPac Compatible | |

DS-CS202*E-DMX-3-1-R1

**www.arscryo.com**

Instrumentation Receptacle Pin-Out

Work Order #. 16-A221

Controller Type: LS336

PORT A

| Connector Pin | Function (Connection) | |
|---|---|---|
| A | E-TYPE T/C (Chromel+) Controls 700/800K | Input-D |
| B | E-TYPE T/C (Constantan-) Controls 700/800K | Curve-22 |
| C | Open | |
| D | Open | |
| E | Open | |
| F | Open | |
| G | PT-103 used for better Accuracy at high temp.I+ | 40K to 800K |
| H | PT-103 used for better Accuracy at high temp.V+ | Input-C |
| J | PT-103 used for better Accuracy at high temp.I- | Curve-06 |
| K | PT-103 used for better Accuracy at high temp.V- | |

PORT B

| Connector Pin | Funcition (Connection) | |
|---|---|---|
| A | Reference sensor mounted on cold stage I+ | Input -A |
| B | Reference sensor mounted on cold stage V+ | Curve-02 |
| C | Reference sensor mounted on cold stage I- | DT-670B-SD |
| D | Reference sensor mounted on cold stage V- | |
| E | Heater | 50-Ohm |
| F | Heater | Cartridge heater |
| G | Cernox sensor freelength good to 420K only I+ | Input-B |
| H | Cernox sensor freelength good to 420K only V+ | Curve-21 |
| J | Cernox sensor freelength good to 420K only I- | S/N-X119992 |
| K | Cernox sensor freelength good to 420K only V- | CX-1030-SD-HT-1.4M |

700 800K Probe Station
Instrumentation Receptacle Pin-Out
32-PIN

Work Order #. Port-C

Controller Type/Sn. LS336

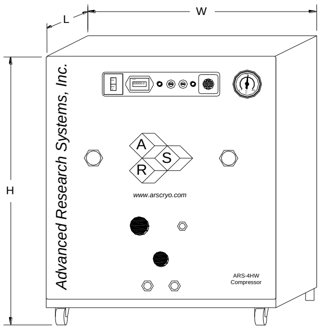
| | Connector Pin | Function (Connection) |
|---|---|---|
| | A | Phosphor Bronze |
| | B | Twisted Pair #1 High Tenp. |
| | C | Phosphor Bronze |
| | D | Twisted Pair #2 High Tenp. |
| | E | Phosphor Bronze |
| | F | Twisted Pair #3 High Temp |
| | G | Phosphor Bronze |
| | H | Twisted Pair #4 High Temp. |
| | J | Phosphor Bronze |
| | K | Twisted Pair #5 High Temp |
| | L | Phosphor Bronze |
| | M | Twisted Pair #6 High Temp. |
| | N | Phosphor Bronze |
| | P | Twisted Pair #7 High Temp. |
| | R | 26 gauge copper wire |
| | S | Twisted Pair Kapton Coated |
| | T | Open |
| | U | Open |
| | V | Open |
| | W | Open |
| | X | Open |
| | Y | Open |
| | Z | Open |
| | a | Open |
| | b | Open |
| | c | Open |
| | d | Open |
| | e | Open |
| | f | Open |
| | g | Open |
| | h | Open |
| | j | Open |

*Advanced Research Systems, Inc.*

## ARS-4HW Specifications

**Dimensions**

Width 17.1 in (434 mm)
Length 19.0 in (483 mm)
Height 20.3 in (516 mm) with casters (standard)
19.4 in (493 mm) with glides (optional)




**Weight**

160 lb (72.6 kg)

**Mounting Position**

Sitting on its casters (or glides) and level within 5°

**Ambient**

Operating: 12-40 C (54-104 F)
with optional air-cooled CoolPac™: < 32 C (90 F)
Storage: -20 to 60 C (-4 to 140 F) with water removed

| CAUTION ! |
| --- |
| **This equipment is for indoor use only.** |

**Electrical Power Requirements**

208-230 VAC ± 5%, 1 Ph, 60 Hz
200 VAC ± 5%, 1 Ph, 50 Hz
19 FLA
80 LRA
25 A MIN. external electrical service rating
30 A MAX. external electrical service circuit breaker or fuse
Nominal 3.7 kVA (3.6 kW) @ 60 Hz
Nominal 3.4 kVA (3.0 kW) @ 50 Hz

Transformers are required for voltages outside the above voltage ranges. Transformers are available from ARS Inc. Typical step-down (buck) transformers are applied as follows:

-10% for nominal 220 VAC, 50 Hz
-10% for nominal 230 VAC, 50 Hz
-15% for nominal 240 VAC, 50 Hz
-20% for nominal 250 VAC, 50 Hz

3

***Advanced Research Systems, Inc.***

## ARS-4HW Specifications

### Cooling Water Requirements

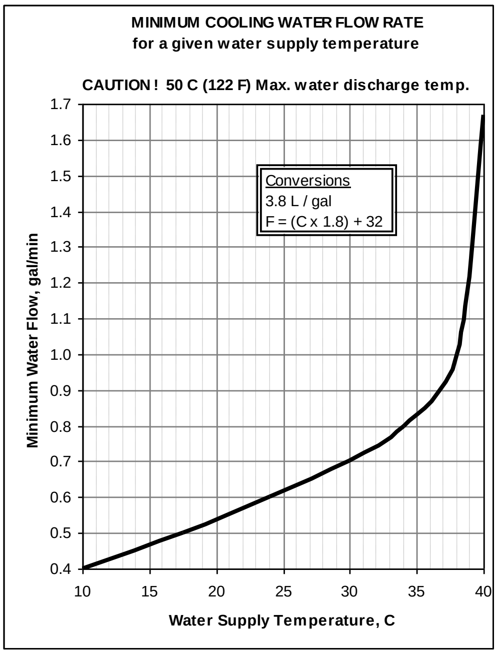
Typical: > 0.6 gal/min (2.3 L/min) with < 24 C (75 F) and > 25 psig (173 kPa) water supply, discharging to drain at < 40 C (100 F)

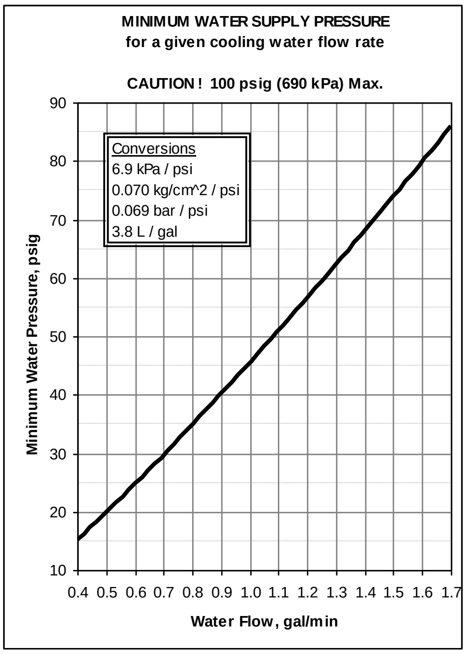
See charts below for minimum flow and pressure requirements:

**CAUTION !**

**Do not exceed 50 C (122 F) MAX. water discharge temperature at compressor.**

**Do not exceed 100 psig (690 kPa) MAX. water supply pressure.**




### Water Quality

Typical municipal drinking water quality is recommended:

pH of 6-8 and total hardness < 85 ppm (5 grains/gal) $CaCO_3$

### Air Cooling (optional)

Use ARS, Inc. CoolPac™

*Advanced Research Systems, Inc.*

## ARS-4HW Specifications

**Refrigerant Gas**

Helium, 99.999% ultra-high purity, with a dew point < -50 C (-58 F) at 300 psig (2069 kPa)

Static pressure: 200-205 psig (1379 -1413 kPa) @ 19-25 C (67-77 F)

Operating supply pressure range: 270 ± 20 psig (1862 ± 138 kPa)

**Interfaces**

- Expander power receptacle: Mates with ARS Inc. standard expander power cable.
- Compressor input power cord: Standard 10 ft (3.0 m) long; universal rated, 300 V, 30 A, 10/3 SJT and HO5VVF3G6; EU-harmonized color code.
- Compressor input power cord plug (for USA and Canada): NEMA L6-30P twist-lock.
- Helium connections: Male self-sealing gas couplings to mate with ARS Inc. flexible gas lines. Valve and 1/4 in ( 6.4 mm) o.d. tube compression fitting for gas fill/vent.
- Water connections: 3/8 in (9.5 mm) o.d. tube compression fittings (polyethylene tubing provided: 40 ft (12 m) length, 190 psi (1310 kPa) working pressure rating @ 24 C (75 F)).
- Elapsed Time Meter (ETM): Displays total time unit has operated when power is applied.

**Safety**

- 22-25 A On/Off Switch-Circuit Breaker, with green indicator light
- Fused controls circuit (F1): 2 A, 250 V, type 3AG (¼ in o.d. x 1¼ in long), quick-acting
- Fused expander power (F2): 1 A, 250 V, type 3AG (¼ in o.d. x 1¼ in long), quick-acting, with green indicator light on front panel
- High Temperature Switch (HTS) with red Over Temp indicator light on front panel; automatically resets
- Compressor motor internal over-current/temperature switch; automatically resets
- Gas supply pressure gauge
- Gas bypass Internal Relief Valve (IRV) and Equalization Solenoid Valve (ESV)
- Atmospheric Relief Valve (ARV) set at 350 psig (2410 kPa), ASME certified ± 3%
- Pressure vessels designed to ASME code Section VIII Division I (although exempt from requiring ASME stamp due to size), and PED 97/23/EC (Group 2 gas, Category I, Module A); 400 psi (2760 kPa) design pressure, 500 psi (3450 kPa) pneumatic proof pressure
- Electrical components rated UL, CSA, CE; Wiring designed to NFPA 79 and LVD 73/23/EEC; Insulation co-ordination per EN61010-1 (Pollution degree 1, Installation category II)
- Enclosure is ~ NEMA/UL/CSA Type 1 (indoor use, protection against contact with internals) and ~ IEC/IP21 (protected from intrusion of solid objects > 12 mm and vertical falling water)

**Scheduled Maintenance**

- Replace adsorber after 12,000 hours of operation

5

## 1.3 Model 336 Specifications

### 1.3.1 Input Specifications

| Standard inputs and scanner option *Model 3062* | Sensor Temperature Coefficient | Input Range | Excitation Current | Display Resolution | Measurement Resolution | Electronic Accuracy (at 25 °C) | Measurement Temperature Coefficient | Electronic Control Stability[8] |
|---|---|---|---|---|---|---|---|---|
| **Diode** | Negative | 0 V to 2.5 V | 10 µA ±0.05%[9,10] | 100 µV | 10 µV | ±80 µV ±0.005% of rdg | (10 µV + 0.0005% of rdg)/°C | ±20 µV |
| | Negative | 0 V to 10 V | 10 µA ±0.05%[9,10] | 100 µV | 20 µV | ±320 µV ±0.01% of rdg | (20 µV + 0.0005% of rdg)/°C | ±40 µV |
| **PTC RTD** | Positive | 0 Ω to 10 Ω | 1 mA[11] | 0.1 mΩ | 0.2 mΩ | ±0.002 Ω ±0.01% of rdg | (0.01 mΩ + 0.001% of rdg)/°C | ±0.4 mΩ |
| | | 0 Ω to 30 Ω | 1 mA[11] | 0.1 mΩ | 0.2 mΩ | ±0.002 Ω ±0.01% of rdg | (0.03 mΩ + 0.001% of rdg)/°C | ±0.4 mΩ |
| | | 0 Ω to 100 Ω | 1 mA[11] | 1 mΩ | 2 mΩ | ±0.004 Ω ±0.01% of rdg | (0.1 mΩ + 0.001% of rdg)/°C | ±4 mΩ |
| | | 0 Ω to 300 Ω | 1 mA[11] | 1 mΩ | 2 mΩ | ±0.004 Ω ±0.01% of rdg | (0.3 mΩ + 0.001% of rdg)/°C | ±4 mΩ |
| | | 0 Ω to 1 kΩ | 1 mA[11] | 10 mΩ | 20 mΩ | ±0.04 Ω ±0.02% of rdg | (1 mΩ + 0.001% of rdg)/°C | ±40 mΩ |
| | | 0 Ω to 3 kΩ | 1 mA[11] | 10 mΩ | 20 mΩ | ±0.04 Ω ±0.02% of rdg | (3 mΩ + 0.001% of rdg)/°C | ±40 mΩ |
| | | 0 Ω to 10 kΩ | 1 mA[11] | 100 mΩ | 200 mΩ | ±0.4 Ω ±0.02% of rdg | (10 mΩ + 0.001% of rdg)/°C | ±400 mΩ |
| **NTC RTD 10 mV** | Negative | 0 Ω to 10 Ω | 1 mA[11] | 0.1 mΩ | 0.2 mΩ | ±0.002Ω ±0.06% of rdg | (0.01 mΩ + 0.001% of rdg)/°C | ±0.3 mΩ |
| | | 0 Ω to 30 Ω | 300 µA[11] | 0.1 mΩ | 0.2 mΩ | ±0.002 Ω ±0.06% of rdg | (0.03 mΩ + 0.001% of rdg)/°C | ±0.9 mΩ |
| | | 0 Ω to 100 Ω | 100 µA[11] | 1 mΩ | 1 mΩ | ±0.01 Ω ±0.04% of rdg | (0.1 mΩ + 0.001% of rdg)/°C | ±3 mΩ |
| | | 0 Ω to 300 Ω | 30µA[11] | 1 mΩ | 2 mΩ | ±0.01 Ω ±0.04% of rdg | (0.3 mΩ + 0.001% of rdg)/°C | ±9 mΩ |
| | | 0 Ω to 1 kΩ | 10 µA[11] | 10 mΩ | 10 mΩ +0.002% of rdg | ±0.1 Ω ±0.04% of rdg | (1 mΩ + 0.001% of rdg)/°C | ±30 mΩ ±0.004% of rdg |
| | | 0 Ω to 3 kΩ | 3 µA[11] | 10 mΩ | 20 mΩ +0.002% of rdg | ±0.1 Ω ±0.04% of rdg | (3 mΩ + 0.001% of rdg)/°C | ±90 mΩ ±0.004% of rdg |
| | | 0 Ω to 10 kΩ | 1 µA[11] | 100 mΩ | 100 mΩ +0.002% of rdg | ±1.0 Ω ±0.04% of rdg | (10 mΩ + 0.001% of rdg)/°C | ±300 mΩ ±0.004% of rdg |
| | | 0 Ω to 30 kΩ | 300 nA[11] | 100 mΩ | 200 mΩ +0.002% of rdg | ±2.0 Ω ±0.04% of rdg | (30 mΩ + 0.001% of rdg)/°C | ±900 mΩ ±0.004% of rdg |
| | | 0Ω to 100 kΩ | 100 nA[11] | 1 Ω | 1Ω +0.005% of rdg | ±10.0 Ω ±0.04% of rdg | (100 mΩ + 0.001% of rdg)/°C | ±3 Ω ±0.01% of rdg |

*[8] Control stability of the electronics only, in ideal thermal system*
*[9] Current source error has negligible effect on measurement accuracy*
*[10] Diode input excitation can be set to 1 mA*
*[11] Current source error is removed during calibration*
*[12] Accuracy specification does not include errors from room temperature compensation*

*TABLE 1-3* ***Input specifications***

| Thermocouple option *Model 3060* | Sensor Temperature Coefficient | Input Range | Excitation Current | Display Resolution | Measurement Resolution | Electronic Accuracy (at 25 °C) | Measurement Temperature Coefficient | Electronic Control Stability[13] |
|---|---|---|---|---|---|---|---|---|
| **Thermocouple 3060** | Positive | ±50 mV | NA | 0.1 µV | 0.4µV | ±1 µV ±0.05% of rdg[12] | (0.1 µV + 0.001% of rdg)/°C | ±0.8µV |

*[13] Control stability of the electronics only, in ideal thermal system*

*TABLE 1-4* ***Thermocouple option input specifications***

| Capacitance option *Model 3061* | Sensor Temperature Coefficient | Input Range | Excitation Current | Display Resolution | Measurement Resolution | Electronic Accuracy (at 25 °C) | Measurement Temperature Coefficient | Electronic Control Stability[14] |
|---|---|---|---|---|---|---|---|---|
| **Capacitance 3061** | Positive or negative | 0.1 nF to 15 nF | 3.496 kHz 1 mA square wave | 0.1 pF | 0.05 pF | ±50 pF ±0.1% of rdg | 2.5 pF/°C | 0.1 pF |
| | | 1 nF to 150 nF | 3.496 kHz 10 mA square wave | 1 pF | 0.5 pF | ±50 pF ±0.1% of rdg | 5 pF/°C | 1 pF |

*[14] Control stability of the electronics only, in ideal thermal system*

*TABLE 1-5* ***Capacitance option input specifications***

## 1.3.2 Sensor Input Configuration

| | Diode/RTD | Thermocouple |
|---|---|---|
| **Measurement type** | 4-lead differential | 2-lead differential, room temperature compensated |
| **Excitation** | Constant current with current reversal for RTDs | NA |
| **Supported sensors** | **Diodes**: Silicon, GaAlAs **RTDs**: 100 Ω Platinum (option), 1000 ) Platinum, Germanium, Carbon-Glass, Cernox™, and Rox™ | Most thermocouple types |
| **Standard curves** | DT-470, DT-670, DT-500-D, DT-500-E1, PT-100, PT-1000, RX-102A, RX-202A | Type E, Type K, Type T, AuFe 0.07% vs. Cr, AuFe 0.03% vs. CR |
| **Input connector** | 6-pin DIN | Screw terminals in a ceramic isothermal block |

*TABLE 1-6* ***Sensor input configuration***

## 1.3.3 Thermometry

| | |
|---|---|
| **Number of inputs** | 4 (8 with Model 3062) |
| **Input configuration** | Inputs can be configured from the front panel to accept any of the supported input types. Thermocouple and capacitance inputs require an optional input card that can be installed in the field. |
| **Supported option cards** | Thermocouple (3060), capacitance (3061), or scanner (3062) |
| **Option slots** | 1 |
| **Isolation** | Sensor inputs optically isolated from other circuits but not each other |
| **A/D resolution** | 24-bit |
| **Input accuracy** | Sensor dependent, refer to Input Specifications table |
| **Measurement resolution** | Sensor dependent, refer to Input Specifications table |
| **Maximum update rate** | 10 rdg/s on each input, 5 rdg/s when configured as 100 kΩ NTC RTD with reversal on |
| **Maximum update rate (scanner)** | The maximum update rate for a scanned input is 10 rdg/s distributed among the enabled channels. Any channel configured as 100 kΩ RTD with reversal on changes the update rate for the channel to 5 rdg/s |
| **Autorange** | Automatically selects appropriate NTC RTD or PTC RTD range |
| **User curves** | Room for 39 200-point CalCurves™ or user curves |
| **SoftCal™** | Improves accuracy of DT-470 diode to ±0.25 K from 30 K to 375 K; improves accuracy of platinum RTDs to ±0.25 K from 70 K to 325 K; stored as user curves |
| **Math** | Maximum and minimum |
| **Filter** | Averages 2 to 64 input readings |

## 1.3.4 Control

There are 4 control outputs.

### 1.3.4.1 Heater Outputs (Outputs 1 and 2)

| | |
|---|---|
| **Control type** | Closed loop digital PID with manual heater output or open loop |
| **Update rate** | 10/s |
| **Tuning** | Autotune (one loop at a time), PID, PID zones |
| **Control stability** | Sensor dependent, see Input Specifications table |
| **PID control settings** | |
| **Proportional (gain)** | 0 to 1000 with 0.1 setting resolution |
| **Integral (reset)** | 1 to 1000 (1000/s) with 0.1 setting resolution |
| **Derivative (rate)** | 1 to 200% with 1% resolution |
| **Manual output** | 0 to 100% with 0.01% setting resolution |
| **Zone control** | 10 temperature zones with P, I, D, manual heater out, heater range, control channel, ramp rate |
| **Setpoint rampin** | 0.1 K/min to 100 K/min |

| | 25 Ω setting | 50 Ω setting |
|---|---|---|
| **Type** | Variable DC current source | |
| **D/A resolution** | 16-bit | |
| **Max power** | 100 W | 50 W |
| **Max current** | 2 A | 1 A |
| **Compliance voltage** | 50 V | 50 V |
| **Heater load for max power** | 25 Ω | 50 Ω |
| **Heater load range** | 10 Ω to 100 Ω | |
| **Ranges** | 3 (decade steps in power) | |
| **Heater noise** | 0.12 µA RMS (dominated by line frequency and its harmonics) | |
| **Grounding** | Output referenced to chassis ground | |
| **Heater connector** | Dual banana | |
| **Safety limits** | Curve temperature, power up heater off, short circuit protection | |

*TABLE 1-7* ***Output 1***

| | 25 Ω setting | 50 Ω setting |
|---|---|---|
| **Type** | Variable DC current source | |
| **D/A resolution** | 16-bit | |
| **Max power** | 50 W | 50 W |
| **Max current** | 1.41 A | 1 A |
| **Compliance voltage** | 35.4 V | 50 V |
| **Heater load for max power** | 25 Ω | 50 Ω |
| **Heater load range** | 10 Ω to 100 Ω | |
| **Ranges** | 3 (decade steps in power) | |
| **Heater noise** | 0.12 µA RMS (dominated by line frequency and its harmonics) | |
| **Grounding** | Output referenced to chassis ground | |
| **Heater connector** | Dual banana | |
| **Safety limits** | Curve temperature, power up heater off, short circuit protection | |

*TABLE 1-8* ***Output 2***

### 1.3.4.2 Unpowered Analog Outputs (Outputs 3 and 4)

| | |
|---|---|
| **Control type** | Closed loop PID, PID zones, warm up heater mode, manual output or Monitor Out |
| **Tuning** | Autotune (one loop at a time), PID, PID zones |
| **Control stability** | Sensor dependedn, see Input Specifications table |
| **PID control settings** | |
| **Proportional (gain)** | 0 to 1000 with 0.1 setting resolution |
| **Integral (reset)** | 1 to 1000 (1000/s) with 0.1 setting resolution |
| **Derivative (rate)** | 1 to 200% with 1% resolution |
| **Manual output** | 0 to 100% with 0.01% setting resolution |
| **Zone control** | 10 temperature zones with P, I, D, manual heater out, heater range, control channel, ramp rate |
| **Setpoint ramping** | 0.1 K/min to 100 K/min |
| **Warm up heater mode settings** | |
| **Warm up percentage** | 0 to 100% with 1% resolution |
| **Warm up mode** | Continuous control or auto-off |
| **Monitor Out settings** | |
| **Scale** | User selected |
| **Data source** | Temperature or sensor units |
| **Settings** | Input, source, top of scale, bottom of scale or manual |
| **Type** | Variable DC voltage source |
| **Update rate** | 10/s |
| **Range** | ±10 V |
| **Resolution** | 16-bit, 0.3 mV |
| **Accuracy** | ±2.5 mV |
| **Noise** | 0.3 mV RMS |
| **Minimum load resistance** | 1 kΩ (short-circuit protected) |
| **Connector** | Detachable terminal block |

## 1.3.5 Front Panel

| | |
|---|---|
| **Display** | 8-line by 40-character (240 × 64 pixel) graphic LCD display module with LED backlight |
| **Number of reading displays** | 1 to 8 |
| **Display units** | K, °C, V, mV, Ω |
| **Reading source** | Temperature, sensor units, max, and min |
| **Display update rate** | 2 rdg/s |
| **Temperature display resolution** | 0.0001° from 0° to 99.9999°, 0.001° from 100° to 999.999°, 0.01° above 1000° |
| **Sensor units display resolution** | Sensor dependent, to 6 digits |
| **Other displays** | Input name, setpoint, heater range, heater output, and PID |
| **Setpoint setting resolution** | Same as display resolution (actual resolution is sensor dependent) |
| **Heater output display** | Numeric display in percent of full scale for power or current |
| **Heater output resolution** | 0.01% |
| **Display annunciators** | Control input, alarm, tuning |
| **LED annunciators** | Remote, Ethernet status, alarm, control outputs |
| **Keypad** | 27-key silicone elastomer keypad |
| **Front panel features** | Front panel curve entry, display contrast control, and keypad lock-out |

# Appendix A. Appendix



## 1.3.6 Interface

| | |
|---|---|
| **IEEE-488.2** | |
| **Capabilities** | SH1, AH1, T5, L4, SR1, RL1, PP0, DC1, DT0, C0, E1 |
| **Reading rate** | To 10 rdg/s on each input |
| **Software support** | LabVIEW™ driver (contact Lake Shore for availability) |
| **USB** | |
| **Function** | Emulates a standard RS-232 serial port |
| **Baud Rate** | 57,600 |
| **Connector** | B-type USB connector |
| **Reading rate** | To 10 rdg/s on each input |
| **Software support** | LabVIEW™ driver (contact Lake Shore for availability) |
| **Ethernet** | |
| **Function** | TCP/IP web interface, curve handler, configuration backup, chart recorder |
| **Connector** | RJ-45 |
| **Reading rate** | To 10 rdg/s on each input |
| **Software support** | LabVIEW™ driver (contact Lake Shore for availability) |
| **Alarms** | |
| **Number** | 4, high and low for each input |
| **Data source** | Temperature or sensor units |
| **Settings** | Source, high setpoint, low setpoint, deadband, latching or non-latching, audible on/off, and visible on/off |
| **Actuators** | Display annunciator, beeper, and relays |
| **Relays** | |
| **Number** | 2 |
| **Contacts** | Normally open (NO), normally closed (NC), and common (C) |
| **Contact rating** | 30 VDC at 3 A |
| **Operation** | Activate relays on high, low, or both alarms for any input, or manual mode |
| **Connector** | Detachable terminal block |

## 1.3.7 General

| | |
|---|---|
| **Ambient temperature** | 15 °C to 35 °C at rated accuracy; 5 °C to 40 °C at reduced accuracy |
| **Power requirement** | 100, 120, 220, 240, VAC, ±10%, 50 or 60 Hz, 250 VA |
| **Size** | 435 mm W × 89 mm H × 368 mm D (17 in × 3.5 in × 14.5 in), full rack |
| **Weight** | 7.6 kg (16.8 lb) |
| **Approval** | CE mark |

## A.1.4. Keithley SourceMeter 2600

**Manufacturer:** Keithley Instruments, Inc.
**Type:** SourceMeter 2614B
**Serial:** 4038238
**TUGraz inventory number:** 0113863

**Manufacturer:** Keithley Instruments, Inc.
**Type:** SourceMeter 2636A
**Serial:** 1239787
**TUGraz inventory number:** 0105311

| Model 2601B/2602B/2604B | | | Model 2611B/2612B/2614B | | | Model 2634B/2635B/2636B | | |
|---|---|---|---|---|---|---|---|---|
| Range | Source | Measure | Range | Source | Measure | Range | Source | Measure |
| 100 mV | ±101 mV | ±102 mV | 200 mV | ±202 mV | ±204 mV | 200 mV | ±202 mV | ±204 mV |
| 1 V | ±1.01 V | ±1.02 V | 2 V | ±2.02 V | ±2.04 V | 2 V | ±2.02 V | ±2.04 V |
| 6 V | ±6.06 V | ±6.12 V | 20 V | ±20.2 V | ±20.4 V | 20 V | ±20.2 V | ±20.4 V |
| 40 V | ±40.4 V | ±40.8 V | 200 V[1] | ±202 V | ±204 V | 200 V[3] | ±202 V | ±204 V |
| 100 nA | ±101 nA | ±102 nA | 100 nA | ±101 nA | ±102 nA | 100 pA[4] | N/A | ±102 pA |
| 1 µA | ±1.01 µA | ±1.02 µA | 1 µA | ±1.01 µA | ±1.02 µA | 1 nA | ±1.01 nA | ±1.02 nA |
| 10 µA | ±10.1 µA | ±10.2 µA | 10 µA | ±10.1 µA | ±10.2 µA | 10 nA | ±10.1 nA | ±10.2 nA |
| 100 µA | ±101 µA | ±102 µA | 100 µA | ±101 µA | ±102 µA | 100 nA | ±101 nA | ±102 nA |
| 1 mA | ±1.01 mA | ±1.02 mA | 1 mA | ±1.01 mA | ±1.02 mA | 1 µA | ±1.01 µA | ±1.02 µA |
| 10 mA | ±10.1 mA | ±10.2 mA | 10 mA | ±10.1 mA | ±10.2 mA | 10 µA | ±10.1 µA | ±10.2 µA |
| 100 mA | ±101 mA | ±102 mA | 100 mA | ±101 mA | ±102 mA | 100 µA | ±101 µA | ±102 µA |
| 1 A | ±1.01 A | ±1.02 A | 1 A | ±1.01 A | ±1.02 A | 1 mA | ±1.01 mA | ±1.02 mA |
| 3 A | ±3.03 A | ±3.06 A | 1.5 A | ±1.515 A | ±1.53 A | 10 mA | ±10.1 mA | ±10.2 mA |
| | | | 10 A[2] | ±10.1 A | ±10.2 A | 100 mA | ±101 mA | ±102 mA |
| | | | | | | 1 A | ±1.01 A | ±1.02 A |
| | | | | | | 1.5 A | ±1.515 A | ±1.53 A |
| Max Power = 40.4 W per channel | | | Max Power = 30.603 W per channel<br>1. 200 V source range available only when interlock is enabled. See Digital I/O (on page 3-83).<br>2. 10 A range available only in pulse mode. | | | Max Power = 30.603 W per channel<br>3. 200 V source range available only when interlock is enabled. See Digital I/O (on page 3-83).<br>4. 100 pA range is not available on the Model 2634B. | | |

Figure A.5.: The table lists the source and measure limits for the voltage and current functions of the Keithley SMU2600 series [22]

## A.1.5. Agilent Switch Mainframe

**Manufacturer:** Agilent Technologies, Inc.
**Type:** 3499A
**Serial:** CN40053425

# 44473A 4 x 4 2-Wire Matrix Switch Module

| | | |
|---|---|---|
| **■ INPUT CHARACTERISTICS** | | |
| **Total Channels:** | 16 | |
| **Maximum Voltage:** | Terminal-Terminal or Terminal-Chassis: | 250 V, dc or ac rms |
| **Maximum Current:** | Per Channel:<br>Per Module: | 2 A, dc or ac rms<br>8 A, dc or ac rms |
| **Maximum Power:** | Per Channel:<br>Per Module: | 60 W dc; 500 VA ac<br>240 W dc; 2000 VA ac |
| **Maximum Overvoltage Transients:** | 1400 $V_{pk}$ | |
| **Thermal Offset:** | < 3 µV differential | |
| **Initial Closed Channel Resistance:** | < 1 Ω | |
| **Relay Life:** | Dry Load of < 300 mA & < 10 V:<br>Maximum Rated Load: | $10^8$<br>$10^5$ |
| **Maximum Scan Rate:**[a] | 43 Chans/sec | |
| **■ DC ISOLATION** | | |
| **Open Channel, Channel-Channel:** (with 1 channel closed) | < (40°C, 60% RH):<br>< (40°C, 95% RH): | > $10^{11}$ Ω<br>> $10^9$ Ω |
| **HI-LO:** (with 1 channel closed) | < (40°C, 60% RH):<br>< (40°C, 95% RH): | > $10^{10}$ Ω<br>> $10^8$ Ω |
| **Channel-Chassis:** (with 1 channel closed) | < (40°C, 60% RH):<br>< (40°C, 95% RH): | > $10^{10}$ Ω<br>> $5x10^8$ Ω |

a. Using the 44474A external increment & channel closed, display off.

| ■ AC ISOLATION / PERFORMANCE[a] | | |
|---|---|---|
| **Capacitance:**<br>(with 1 channel closed) | Open Channel, Channel-Channel:<br>HI-LO:<br>Channel-Chassis: | < 5 pF<br>< 40 pF<br>< 70 pF |
| **Insertion Loss:**<br>(with 50Ω termination) | 100 kHz:<br>1 MHz:<br>10 MHz: | < 0.30 dB<br>< 0.35 dB<br>< 0.90 dB |
| **Crosstalk:**<br>(with 50Ω termination) | 100 kHz:<br>1 MHz:<br>10 MHz: | < -76 dB<br>< -56 dB<br>< -36 dB |

a. With chassis of all instruments connected, and with the Lo of input lines connected to the Lo of output lines (either directly or via the 3499A/B/C switching channels).

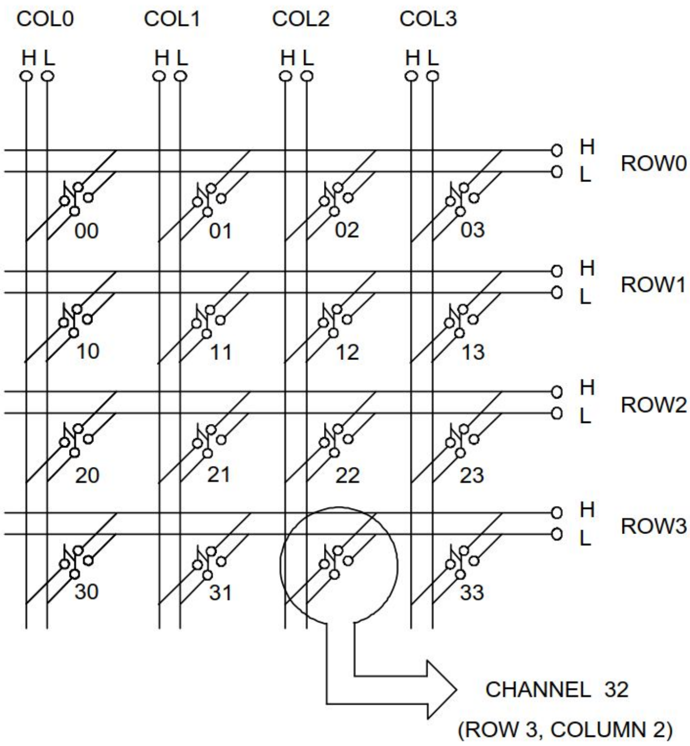



Figure A.6.: A simplified schematic of the 44473A 4 x 4 2-Wire Matrix Switch Module. It consists of 16 2-wire relays (nodes/crosspoints) organized in a 4-row by 4-column matrix [33].

## A.1.6. Magnet-Physic magnetometer

**Manufacturer:** Magnet-Physik Dr. Steingroever GmbH
**Type:** FH 54
**Serial:** 122310
**TUGraz inventory number:** 0190635

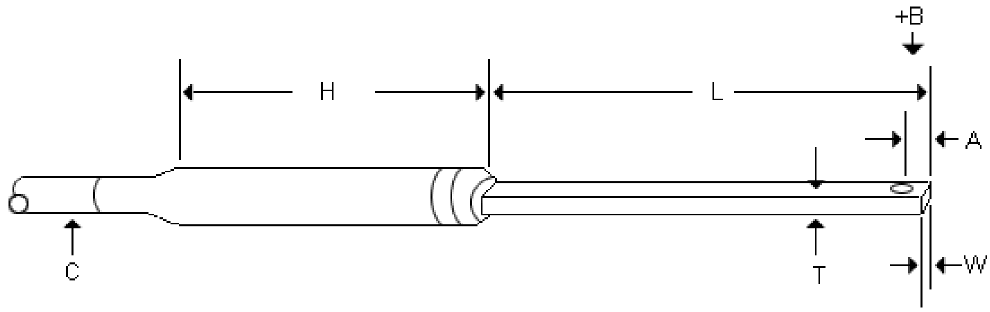
| Modell | FH 54 | | |
|---|---|---|---|
| Anzeige | 3¾-stellig (0...±2999) | | |
| Einheiten | Tesla, Gauss, Ampere pro Meter | | |
| Messbereiche | 30 µT* | 300 mG* | 24 A/m* |
| | 300 µT* | 3 G* | 240 A/m* |
| | 3 mT | 30 G | 2,4 kA/m |
| | 30 mT | 300 G | 24 kA/m |
| | 300 mT | 3 kG | 240 kA/m |
| | 3 T | 30 kG | 2,4 MA/m |
| | 30 T* | 300 kG* | 24 MA/m* |
| | *besondere Sonden erforderlich | | |
| Auflösung (im empfindlichsten Bereich) | abhängig vom Sondentyp | | |
| Frequenzbereich | DC (mit Polaritätsanzeige)<br>AC ca. 20 Hz - 20 kHz (Effektivwert, Grenzen abhängig von der Aussteuerung und vom Sondentyp) | | |
| Grundgenauigkeit<br>Vergleichspräzision | DC: 0,3 %, AC: 2 % (ohne Sonde)<br>DC: 0,2 %, AC: 1 % (ohne Sonde) | | |
| Spitzenwertspeicher (Peak Hold) | Anstiegszeit des Impulses > 150 µs | | |
| Analogausgang | ± 3 V, BNC Anschluss | | |
| Computer-Schnittstelle | RS 232, DB-9 Anschluss | | |
| Temperaturbereich<br>- Betrieb<br>- Lagerung | <br>+10 °C to +40 °C<br>-40 °C to +60 °C | | |
| Stromversorgung<br>- Betriebsdauer mit Batterien | Batterien, 5 Stück 1,5 V, Größe AA (LR6), oder über Steckernetzteil<br>abhängig vom Sondentyp | | |
| Zubehör/Optionen:<br>- Hall-Sonden<br>- Sondenanschlusskabel<br>- Magnetische Abschirmkammer<br>- Aufbewahrungskoffer<br>- Steckernetzteil<br>- Schutzhülle mit Schultergurt<br>- Relaisausgang für Grenzwerte | <br>verschiedene, siehe Sondenkatalog<br>fest mit der Sonde verbunden, verschiedene Längen erhältlich<br>im Lieferumfang<br>im Lieferumfang<br>optional, für Dauerbetrieb empfohlen, verschiedene Netzsteckertypen verfügbar<br>optional, Überzug zum Schutz des Geräts vor Stößen, aus gummiartigem Material<br>optional, 2 Wechsler | | |
| Außenabmessungen | 266 mm x 90 / 144 mm x 60 mm | | |
| Gewicht | ca. 0,5 kg | | |

Figure A.7.: Technical specifications of the Magnet-Physik FH 54 magnetometer [34].

MAGNET-PHYSIK
Dr. Steingroever GmbH
www.magnet-physik.de
Seite 1 / 2

# HALL-SONDEN FÜR FH 54 UND FH 55

## • Transversale Hall-Sonden für FH 54 und FH 55




**Transversalsonden für FH 54 und FH 55**

| Modell | HS-TGB5-104005 | HS-TGB5-104010 | HS-TGB5-104020 |
|---|---|---|---|
| W | 4,0 mm max. | 4,0 mm max. | 4,0 mm max. |
| T (max.) | 1,0 mm | 1,0 mm | 1,0 mm |
| L (nom.) | 55 mm | 100 mm | 200 mm |
| A (nom.) | 2 mm ± 0,1 mm | 2 mm ± 0,1 mm | 2 mm ± 0,1 mm |
| H (nom.) | 70 mm | 70 mm | 70 mm |
| Kabellänge C | 1,5 m | 1,5 m | 1,5 m |
| Stabmaterial | Glasfaser-Kunststoff | | |
| Aktive Fläche, nomineller Durchmesser | 0,4 mm | 0,4 mm | 0,4 mm |
| Messbereiche, Vollausschlag | 3 mT bis 3 T | | |
| Korrigierte Genauigkeit [% vom Messwert, DC] | 0,25 % bis 3 T | 0,25 % bis 3 T | 0,25 % bis 3 T |
| Temperaturkoeffizient der Empfindlichkeit (maximal) [% / °C] | ± 0,02 (T) | ± 0,02 (T) | ± 0,02 (T) |

(T): Sonde mit Sensor zur Temperaturkorrektur
Betriebstemperaturbereich 0 °C bis 75 °C.

Aufgrund kontinuierlicher Produktverbesserungen können sich die Spezifikationen ohne Mitteilung ändern.

### A.1.7. Lock-In Amplifiers

Technical specifications are taken form the device manuals [21], [35].

**Stanford Research Systems SR830**

> **Manufacturer:** Stanford Research Systems
> **Type:** SR830
> **Serial:** 83104
> **TUGraz inventory number:** 0067883

**Princeton Applied Research Model 5210**

> **Manufacturer:** Princeton Applied Research
> **Type:** Model 5210
> **Serial:** 05120
> **TUGraz inventory number:** 9528205

## SR830 DSP LOCK-IN AMPLIFIER

# SPECIFICATIONS

| | |
|---|---|
| **SIGNAL CHANNEL** | |
| Voltage Inputs | Single-ended (A) or differential (A-B). |
| Current Input | $10^6$ or $10^8$ Volts/Amp. |
| Full Scale Sensitivity | 2 nV to 1 V in a 1-2-5-10 sequence (expand off). |
| Input Impedance | Voltage: 10 MΩ+25 pF, AC or DC coupled.<br>Current: 1 kΩ to virtual ground. |
| Gain Accuracy | ±1% from 20°C to 30°C (notch filters off), ±0.2 % Typical. |
| Input Noise | 6 nV/√Hz at 1 kHz (typical). |
| Signal Filters | 60 (50) Hz and 120(100) Hz notch filters (Q=4). |
| CMRR | 100 dB to10 kHz (DC Coupled), decreasing by 6db/octave above 10 kHz |
| Dynamic Reserve | Greater than 100 dB (with no signal filters). |
| Harmonic Distortion | -80 dB. |
| **REFERENCE CHANNEL** | |
| Frequency Range | 1 mHz to 102 kHz |
| Reference Input | TTL (rising or falling edge) or Sine.<br>Sine input is1 MΩ, AC coupled (>1 Hz). 400 mV pk-pk minimum signal. |
| Phase Resolution | 0.01° |
| Absolute Phase Error | <1° |
| Relative Phase Error | <0.01° |
| Orthogonality | 90° ± 0.001° |
| Phase Noise | External synthesized reference: 0.005° rms at 1 kHz, 100 ms, 12 dB/oct.<br>Internal reference: crystal synthesized, <0.0001° rms at 1 kHz. |
| Phase Drift | <0.01°/°C below 10 kHz<br><0.1°/°C to 100 kHz |
| Harmonic Detect | Detect at Nxf where N<19999 and Nxf<102 kHz. |
| Acquisition Time | (2 cycles + 5 ms) or 40 ms, whichever is greater. |
| **DEMODULATOR** | |
| Zero Stability | Digital displays have no zero drift on all dynamic reserves.<br>Analog outputs: <5 ppm/°C for all dynamic reserves. |
| Time Constants | 10 µs to 30 s (reference > 200 Hz). 6, 12, 18, 24 dB/oct rolloff.<br>up to 30000 s (reference < 200 Hz). 6, 12, 18, 24 dB/oct rolloff.<br>Synchronous filtering available below 200 Hz. |
| Harmonic Rejection | -80 dB |
| **INTERNAL OSCILLATOR** | |
| Frequency | 1 mHz to 102 kHz. |
| Frequency Accuracy | 25 ppm + 30 µHz |
| Frequency Resolution | 4 1/2 digits or 0.1 mHz, whichever is greater. |
| Distortion | f<10 kHz, below -80 dBc. f>10 kHz, below -70 dBc.1 Vrms amplitude. |
| Output Impedance | 50 Ω |
| Amplitude | 4 mVrms to 5 Vrms (into a high impedance load) with 2 mV resolution.<br>(2 mVrms to 2.5 Vrms into 50Ω load). |
| Amplitude Accuracy | 1% |
| Amplitude Stability | 50 ppm/°C |
| Outputs | Sine output on front panel. TTL sync output on rear panel.<br>When using an external reference, both outputs are phase locked to the external reference. |

*1-5*

## SR830 DSP Lock-In Amplifier

| | |
|---|---|
| **DISPLAYS** | |
| Channel 1 | 4 1/2 digit LED display with 40 segment LED bar graph. X, R, X Noise, Aux Input 1 or 2. The display can also be any of these quantities divided by Aux Input 1 or 2. |
| Channel 2 | 4 1/2 digit LED display with 40 segment LED bar graph. Y, θ, Y Noise, Aux Input 3 or 4. The display can also be any of these quantities divided by Aux Input 3 or 4. |
| Offset | X, Y and R may be offset up to ±105% of full scale. |
| Expand | X, Y and R may be expanded by 10 or 100. |
| Reference | 4 1/2 digit LED display. Display and modify reference frequency or phase, sine output amplitude, harmonic detect, offset percentage (X, Y or R), or Aux Outputs 1-4. |
| Data Buffer | 16k points from both Channel 1 and Channel 2 display may be stored internally. The internal data sample rate ranges from 512 Hz down to 1 point every 16 seconds. Samples can also be externally triggered. The data buffer is accessible only over the computer interface. |
| **INPUTS AND OUTPUTS** | |
| Channel 1 Output | Output proportional to Channel 1 display, or X. Output Voltage: ±10 V full scale. 10 mA max output current. |
| Channel 2 Output | Output proportional to Channel 2 display, or Y. Output Voltage: ±10 V full scale. 10 mA max output current. |
| X and Y Outputs | Rear panel outputs of cosine (X) and sine (Y) components. Output Voltage: ±10 V full scale. 10 mA max output current. |
| Aux. Outputs | 4 BNC Digital to Analog outputs. ±10.5 V full scale, 1 mV resolution. 10 mA max output current. |
| Aux. Inputs | 4 BNC Analog to Digital inputs. Differential inputs with1 MΩ input impedance on both shield and center conductor. ±10.5 V full scale, 1 mV resolution. |
| Trigger Input | TTL trigger input triggers stored data samples. |
| Monitor Output | Analog output of signal amplifiers (before the demodulator). |
| **GENERAL** | |
| Interfaces | IEEE-488 and RS232 interfaces standard. All instrument functions can be controlled through the IEEE-488 and RS232 interfaces. |
| Preamp Power | Power connector for SR550 and SR552 preamplifiers. |
| Power | 40 Watts, 100/120/220/240 VAC, 50/60 Hz. |
| Dimensions | 17"W x 5.25"H x 19.5"D |
| Weight | 30 lbs. |
| Warranty | One year parts and labor on materials and workmanship. |

*1-6*

# Specifications

Appendix A

## Measurement Modes

| | |
|---|---|
| **XY %** | X-channel and Y-channel PSD outputs expressed as a percentage of the present full-scale sensitivity setting |
| **XY V** | X-channel and Y-channel PSD outputs expressed directly in terms of voltage at input to signal channel |
| **R θ** | Vector magnitude of input signal in volts and phase angle in degrees |
| **NOISE** | Noise in a bandwidth defined by the output filter time constant and slope controls and centered at the reference frequency expressed as a percentage of the present full-scale sensitivity setting |
| **Harmonic** | Fundamental (F) or 2F modes |

## Displays & Indicators

Two, 3½ -digit liquid crystal displays, analog center-zero panel meter and back-lit LED indicators show the settings of all the main instrument controls and outputs.

## Signal Channel

| | |
|---|---|
| **Voltage Inputs** | |
| Modes | A only or Differential (A-B) |
| Full-scale Sensitivity | 100 nV to 3 V rms in a 1-3-10 sequence |
| Impedance | 100 MΩ // 30 pF |
| Maximum Input | ± 100 V DC; 30 V AC pk-pk without damage, 10 V AC pk-pk without saturation |
| Voltage Noise | 5 nV/√Hz at 1 kHz typ |
| CMRR | > 100 dB at 1 kHz degrading by 6 dB/octave |
| Frequency Response | 0.5 Hz to 120 kHz |
| Grounding | BNC shields can be grounded or floated via 1 kΩ to ground |
| | |
| **Current Input** | |
| Mode | $10^6$ V/A or $10^8$ V/A |
| Full-scale Sensitivity | |
| $10^8$ V/A | 10 fA to 30 nA in a 1-3-10 sequence |
| $10^6$ V/A | 10 fA to 3 µA in a 1-3-10 sequence |
| Frequency Response | |
| $10^8$ V/A | -3 dB at 330 Hz |
| $10^6$ V/A | -3 dB at 60 kHz |
| Impedance | |
| $10^8$ V/A | < 2.5 kΩ at 100 Hz |
| $10^6$ V/A | < 250 Ω at 1 kHz |

| | |
|---|---|
| Maximum Input | 15 mA continuous, 1 A momentary without damage. 10 µA AC pk-pk without saturation on $10^6$ V/A; 100 nA AC pk-pk without saturation on $10^8$ V/A |
| Noise | |
| $10^8$ V/A | 13 fA/√Hz at 500 Hz |
| $10^6$ V/A | 130 fA/√Hz at 1 kHz |
| Grounding | BNC shield can be grounded or floated via 1 kΩ to ground |
| **Line Notch Filter** | > 34dB attenuation @ ±1% of 50 or 60 Hz and/or 100 or 120 Hz |
| **Dynamic Reserve** | 130 dB max |
| **Gain Accuracy** | |
| Flat Mode | 1% typical |
| Bandpass Mode | 2% typical |
| **Gain Stability** | 200ppm/°C typical |

## Reference Channel

| | |
|---|---|
| **TTL Input (rear panel)** | |
| Frequency Range | 0.5 Hz to 120 kHz |
| **Analog Input (front panel)** | |
| Impedance | 1 MΩ // 30 pF |
| Frequency Range | 0.5 Hz to 120 kHz |
| Level | |
| Sinusoidal Input | 1.0 V rms** |
| Squarewave Input | 100 mV rms** |
| | **Note: Lower levels can be used with the analog input at the expense of increased phase errors. |
| Maximum input voltage | 5.0 V rms |
| **Phase** | |
| Set Resolution | 0.1° or 0.005° increments |
| Accuracy | ±1° typical |
| Noise | 0.005°rms at 100 ms TC, 12 dB/octave |
| Orthogonality | |
| Above 5Hz | 90° ±0.5° |
| 0.5Hz - 5Hz | 90° ±5° max |
| Drift (Flat Mode) | < 0.05°/°C |
| **Lock Acquisition Time** | 2 cycles + 100 ms |

## Demodulator

| | |
|---|---|
| **Description** | Switching type demodulators operating in either square wave or Walsh function modes. |

| | |
|---|---|
| **Output Zero Stability** | |
| High Dynamic Reserve | 500 ppm/ºC |
| Normal | 50 ppm/ºC |
| High Stability | 5 ppm/ºC |
| **Harmonic Rejection** | |
| Low-Pass | >80dB at 1 kHz |
| Bandpass | >60dB at 1 kHz |
| **Time Constants** | |
| Main outputs | 1 ms to 3 ks in a 1-3-10 sequence |
| Roll-off | 6 and 12 dB/octave |
| P.S.D. Monitor Outputs | 100 µs nominal, X-output only |
| Roll-off | 6 dB/octave only |
| **Offset** | Auto and Manual on X and/or Y: ±150 % FS |

## Oscillator

| | |
|---|---|
| **Frequency** | |
| Range | 0.5 Hz to 120 kHz |
| Setting Resolution | better than 1% |
| Absolute Accuracy | ±2% |
| **Distortion (THD)** | 0.5% |
| **Amplitude** | |
| Range | |
| Front panel | 1 mV to 1.999 V |
| Computer Control | 1 mV to 2.000 V and 5.000 V |
| Setting Resolution | |
| 1 mV to 500 mV | 1 mV |
| 501 mV to 2 V | 4 mV |
| Accuracy | |
| 0.001 Hz to 60 kHz | ±0.3 % |
| 60 kHz to 250 kHz | ±0.5 % |
| Stability | 50 ppm/ºC |
| **Output** | |
| Impedance | 900 Ω |

## Auxiliary Inputs

| | |
|---|---|
| **AUX ADC INPUT CH1 - CH4** | |
| Maximum Input | ±15 V |
| Resolution | 1 mV |
| Input Impedance | 1 MΩ // 30 pF |
| Sample Rate | |
| CH1 only | 200 Hz max. |
| CH1 - CH4 | 50 Hz max. |

| | |
|---|---|
| Trigger Mode | Internal or External |
| Trigger input | TTL compatible |

## Outputs

| | |
|---|---|
| **CH1, CH2 Analog Outputs** | |
| Function | X, Y, R, θ, Noise, Ratio and Log Ratio. |
| Amplitude | ±15 V (±10.0 V = ± full scale) |
| Impedance | 1 kΩ |
| **Signal Monitor** | |
| Amplitude | ±10 V max |
| Impedance | 1 kΩ |
| **Aux D/A Output** | |
| Maximum Output | ±15 V |
| Resolution | 1 mV |
| Output Impedance | < 150 Ω |
| **Reference Output** | |
| Waveform | 0 to 5 V square wave |
| Impedance | TTL compatible |
| **Power - Low Voltage** | ±15 V at 100 mA rear panel DIN connector for powering ***SIGNAL RECOVERY*** preamplifiers |

## Interfaces

RS232 and GPIB (IEEE-488). All settings can be adjusted from the front-panel

## General

| | |
|---|---|
| **Power Requirements** | |
| Voltage | 110/120/220/240 VAC |
| Frequency | 50/60 Hz |
| Power | < 130 VA |
| **Dimensions** | |
| Width | 440 mm (17.25") |
| Depth | 89 mm (16.5 ") |
| Height | |
| With feet | 105 mm (4.1 ") |
| Without feet | 89 mm (3.5 ") |
| **Weight** | 9.1 kg (20 lbs) |

*All specifications subject to change without notification*

### A.1.8. Function generator

Technical specifications are taken form the device manual [36].

**Manufacturer:** Philips Industrial & Electro-acoustic Systems
**Type:** PM 5193
**Serial:** LO 593615
**TUGraz inventory number:** 9000452

E 1-2

## 1.2. CHARACTERISTICS

### 1.2.1. Safety Characteristics

This apparatus has been designed and tested in accordance with safety class I requirements of IEC-Publication 348, Safety Requirements for Electronic Measuring Apparatus, and has been supplied in a safe condition. This manual contains some information and warnings which must be followed by the user to ensure safe operation and to retain the apparatus in a safe condition.

### 1.2.2. Performance Characteristics, Specifications

Properties expressed in numerical values with stated tolerance are guaranteed by the manufacturer. Specified non-tolerance values indicate those that could be nominally expected from the mean of a range of identical instruments.
These specifications are valid after a warming-up time of 30 minutes (reference temperature 23° C) and for a termination of the signal output with 50 Ohm.
If not stated otherwise, relative tolerances relate to the set value.

### 1.2.3. Frequency

| | | |
|---|---|---|
| **frequency range** | 0.1 mHz – 50 MHz | depending on function and wave form |
| setting range | | |
| – sine wave | 0.1 mHz – 50 MHz | |
| – square wave | 0.1 mHz – 20 MHz | |
| – pos. pulses | 0.1 mHz – 50 MHz | |
| – neg. pulses | 0.1 mHz – 50 MHz | |
| – triangular wave | 0.1 mHz – 200 kHz | |
| – haversine | 0.1 mHz – 50 kHz | |
| – pos. sawtooth | 0.1 mHz – 20 kHz | |
| – neg. sawtooth | 0.1 mHz – 20 kHz | |
| **setting** | | numerical keys<br>decimal point key<br>dimension key Hz/kHz<br>step function |
| measuring unit | Hz, kHz | selectable with key Hz/kHz.<br>When controlling via IEC/IEEE bus frequency values can only be entered in Hz. |
| indication | 8-digits | 7-segment LED-display;<br>decimal point free selectable |
| max. resolution | 0.1 mHz | |
| setting error limit | $\pm 1 \times 10^{-6}$ | |
| temperature coefficient | $<$ 0.2 ppm/K | |
| long term drift | $<$ 0.3 ppm<br>within 7 hours | |
| aging | $<$ 1 ppm/year | |
| frequency jitter, residual FM rms | $<$ 0.02 %, $<$ 1200 Hz | f $\geqslant$ 2 MHz<br>LF bandwidth 10 Hz – 20 kHz |
| phase jitter rms | $<$ 3 mrad | f $<$ 2 MHz |
| signal-noise ratio (SNR) | $\geqslant$ 55 dBc | frequency $<$ 2 MHz<br>for a 30 kHz band centred on the carrier excluding ± 1 Hz about the carrier. |

E 1-3

**1.2.4. Signal Output** — BNC-connector OUTPUT at the front plate

| | | |
|---|---|---|
| impedance | 50 Ω | |
| wave forms | sine wave<br>square wave<br>pos. pulses<br>neg. pulses<br>triangular wave<br>haversine<br>pos. sawtooth<br>neg. sawtooth | indication with LEDs in the keys |
| amplitude setting | | numerical keys,<br>decimal point key,<br>step function |
| indication | max. 2 1/2-digits | 7-segment display |
| measuring unit | V | amplitude pp or rms,<br>dc-voltage |
| | dBm | ac-level,<br>indication of the measuring unit with LEDs in the keys |

**1.2.4.1. Sine Wave**

| | | | |
|---|---|---|---|
| frequency range | | 0.1 mHz – 50 MHz | |
| **voltage pp** | | | |
| setting range | | 0 – 20 V | |
| – subranges | I: | 2.1 – 20 V | resolution 0.1 V |
| | II: | 0.21 – 2.00 V | resolution 0.01 V |
| | III: | 0 – 0.200 V | resolution 0.001 V |

error limits of the output voltage pp with 50 Ω termination
(nominal value = 1/2 open circuit voltage)

| subranges of open circuit voltage | | FREQUENCY RANGES 0.1 mHz – 1 Hz | 1 Hz – 200 kHz | 200 kHz – 10 MHz | 10 MHz – 50 MHz |
|---|---|---|---|---|---|
| I | 15.1 – 20.0 V | ± 2.5 % | ± 2.0 % | ± 3.5 % | +6/–12 % (+0.5/–1 dB) |
| | 2.1 – 15.0 V | | (± 0.1 dB) | (± 0.25 dB) | ± 8 % (± 0.5 dB) |
| II | 1.51 – 2.00 V | ± 3 % | ± 2.5 % | ± 4 % | +10/–13 % (+0.7/–1.1 dB) |
| | 0.21 – 1.50 V | | (± 0.1 dB) | (± 0.3 dB) | ± 12 % (± 0.7 dB) |
| III | 0.151 – 0.200 V | ± 3.5 % | ± 3.0 % ± 0.15 mV | ± 5 % ± 0.25 mV | ± 15 % (± 1.2 dB) |
| | 0 – 0.150 V | | (± 0.1 dB ± 0.1 mV) | (± 0.4 dB ± 0.25 mV) | (± 30 % ± 0.25 mV) |

The values in brackets specify the flatness of the amplitude response related to the corresponding lower limit of the frequency subrange.

E 1-4

| | | |
|---|---|---|
| temperature coefficient | < 0.1 %/K | f ⩽ 2.146 MHz |
| | < 0.25 %/K | f < 10 MHz |
| | < 0.45 %/K | generally |
| distortion | | f = 1 Hz – 200 kHz, open circuit voltage > 10 Vpp |
| | < 0.5 % | generally |
| | < 0.35 % | open circuit amplitude < 12 Vpp, subrange I < 1.2 Vpp, subrange II < 0.12 Vpp, subrange III |
| harmonics | < – 31 dBc | open circuit voltage ⩾ 10 mVpp |
| | < – 20 dBc | open circuit voltage < 10 mVpp |
| | < – 37 dBc | open circuit voltage ⩾ 10 mVpp, f ⩽ 10 MHz |
| spurious | < – 40 dBc | open circuit voltage ⩾ 100 mVpp |
| | < – 23 dBc | open circuit voltage ⩾ 10 mVpp |
| | < – 6 dBc | open circuit voltage < 10 mVpp |

**voltage rms open circuit**

| | | | |
|---|---|---|---|
| setting range | | 0 – 7 V | |
| – subranges | I: | 1.1 – 7V | resolution 0.1 V |
| | II: | 0.11 – 1.00 V | resolution 0.01 V |
| | III: | 0 – 0.100 V | resolution 0.001 V |

error limits of output voltage rms with 50 Ω termination
(nominal value = 1/2 open circuit voltage)

| subranges of open circuit voltage | | FREQUENCY RANGES 0.1 mHz – 1 Hz | 1 Hz – 200 kHz | 200 kHz – 10 MHz | 10 MHz – 20 MHz |
|---|---|---|---|---|---|
| I | 5.1 – 7.0 V | ± 3.0 % | ± 2.5 % | ± 4.0 % | + 7 / – 13 % |
| | 1.1 – 5.0 V | ± 3.5 % | ± 3.0 % | ± 4.5 % | + 8 / – 9 % |
| II | 0.51 – 1.00 V | ± 5.5 % | ± 5.0 % | ± 6.0 % | + 11 / – 14 % |
| | 0.11 – 0.50 V | ± 4.0 % | ± 3.5 % | ± 4.0 % | ± 11 % |
| III | 0.051 – 0.100 V | ± 5.5 % | ± 5.0 % | ± 7.5 % | + 11 / – 16 % |
| | 0 – 0.050 V | ± 5.5 % ± 0.1 mV | ± 5.0 % ± 0.1 mV | ± 7.5 % ± 0.1 mV | + 11 / – 18 % ± 0.15 mV |

**level** — with 50 Ω termination

setting range — – 45 . . . + 24 dBm — resolution 1 dB

error limits of the output level dBm

| subranges | | FREQUENCY RANGES 0.1 mHz – 1 Hz | 1 Hz – 200 kHz | 200 kHz – 10 MHz | 10 MHz – 50 MHz |
|---|---|---|---|---|---|
| I | 22 . . . 24 dBm | ± 0.2 dB | ± 0.2 dB | ± 0.3 dB | + 0.6 / – 1.0 dB |
| | 5 . . . 21 dBm | ± 0.4 dB | ± 0.3 dB | ± 0.5 dB | + 0.8 / – 0.9 dB |
| II | 2 . . . 4 dBm | ± 0.3 dB | ± 0.3 dB | ± 0.4 dB | + 1.0 / – 1.3 dB |
| | –15 . . . +1 dBm | ± 0.4 dB | ± 0.4 dB | ± 0.5 dB | ± 1 dB |
| III | –30 . . . – 16 dBm | ± 0.4 dB | ± 0.4 dB | ± 0.6 dB | + 1.0 / – 1.3 dB |
| | –45 . . . – 31 dBm | ± 0.6 dB | ± 0.8 dB | ± 1.2 dB | + 2.0 / – 2.5 dB |

## A.2. Source codes

### A.2.1. Calculation of the van der Pauw geometry correction factor

```
import numpy
import scipy.optimize
import matplotlib.pyplot as plt
import csv

# prepare a csv file the calculated values are stored in
f = open('data\\vdp-correction-factor.csv', 'w')
f_csv = csv.writer(f, delimiter=';', lineterminator='\n')

# write headers into the file
f_csv.writerow(['Q', 'f'])

# generate the Q test vector
Q_vec = numpy.arange(1, 100000, 0.1)
data = []

# calculate f for given Q
for Q in Q_vec:

    # define a function that can be passed to the solver
    def func(f):
        y = f/numpy.log(2) * numpy.arccosh(1/2 * numpy.exp(numpy.log(2)/f)) \
            - ((Q - 1) / (Q + 1))
        return y

    # if Q is within the range of 1 the iteration doesn't work.
    # so we specify the result for that
    sigma = 0.01

    if Q < (1 + sigma):
        f_solution = [1]
    else:
        # calculate a numeric solution
        f_solution = scipy.optimize.fsolve(func, numpy.array(0.99999))

    data.append(f_solution[0])

    # write data to the csv file and flush the buffer
    f_csv.writerow([Q, f_solution[0]])
    f.flush()

# Plot the result
plt.figure(1)
plt.semilogx(Q_vec, data)
plt.grid()
plt.ylabel('f')
plt.xlabel('Q')
plt.show()

# close the csv file
f.close()
```

sources/CalculateCorrectionFactorV3.py

### A.2.2. Van der Pauw measurement using four Source Measure Units

```
"""
This program controls the whole process to measure temperature dependent
resistivity using the Van der Pauw method

Programmed by: Peter Luidolt
Last modified: 2017-02-21
"""

# import some standard libraries
from time import strftime, localtime, sleep
import os
```

```
import logging
import sys
import json

# import self written libraries
from libs.UsefulThings import step_list, pt100_r2t
from libs.voetschV3 import VT4002
from libs.KeithleyV13 import SMU26xx
# import functions to make smu configuration easier
from vdp_measurement import measure_vdp


"""
****************************
PARAMETERS
****************************
"""

'''general parameters'''
# define the log-level you want to see
# this parameter influences what is displayed on the console and what is written to the log file
LOG_LEVEL = logging.DEBUG

'''Van der Pauw measurement parameters'''
# current and compliance voltage we make the Van der Pauw resistivity measurements with
VDP_MEASUREMENT_CURRENT = 100e-3
VDP_COMPLIANCE_VOLTAGE = 20
# time after the SMU is enabled until the measurement value is taken (value in seconds)
VDP_SETTLING_TIME = 1

'''Temperature profile parameters'''
# the temperature we start the measurement with in °C
START_TEMPERATURE = 20
# the temperature we measure up to in °C
END_TEMPERATURE = 100
# steps of the temperature; we will
TEMPERATURE_STEP_SIZE = 10
# defines the time a temperature has to be stable before we start a measurement (value in s)
# the temperature needs to stay within the target temperature +- the allowed deviation (value in °C)
TEMPERATURE_SETTLING_TIME = 60
TEMPERATURE_ALLOWED_DEVIATION = 0.2

'''Parameters needed for data analysis and calculations'''
# sample thickness (in nm)
SAMPLE_THICKNESS = 150


"""
****************************
DEFINITIONS
****************************
"""

# variable that stores the absolute path for the directory in which we will write all our measurement
      data and results
base_path = None


"""
****************************
FUNCTIONS
****************************
"""


def main():

    # import the needed global variables
    global base_path

    # create a directory for this measurement. In it all the data will be stored
    base_path = create_base_path()

    # setup the logging
    configure_logging()

    # connect to the the climate
    clim = connect_climate_chamber()

    # connect and setup the smu used for temperature measurement
    [smu_temp, smua_temp] = setup_smu_temperature()

    # connect to the SMUs used for the Van der Pauw measurement
    [smu_alpha, smu_beta, smu_channel_list] = setup_smu_vdp()

    # define a filename to store the Van der Pauw measurements in
    csv_filename = os.path.join(base_path, "vdp_measurements.csv")
```

```python
    # write all the parameters to a json file for further reference
    store_measurement_parameters("parameters.json")

    # Temperature-Loop
    temperature_step_list = step_list(START_TEMPERATURE, END_TEMPERATURE, TEMPERATURE_STEP_SIZE)
    logging.info("generated temperature step list: " + str(temperature_step_list))

    # enable the climate chamber
    logging.info("Climate chamber has been enabled.")

    for temp in temperature_step_list:
        # set temperature and wait for it to get stable
        # this process may take some time because we need to bring the climate chamber to temperature
        # and then wait that also the sample temperature becomes stable
        go_to_temp(clim, temp, smua_temp)

        # Make a Van der Pauw measurement
        measure_vdp(smu_channel_list,
                    VDP_MEASUREMENT_CURRENT,
                    VDP_COMPLIANCE_VOLTAGE,
                    VDP_SETTLING_TIME,
                    temp,
                    csv_filename,
                    logging,
                    smua_temp)

    # properly shut down and disconnect SMUs
    # generate a list with all SMU channels in it
    all_smu_channels = [smua_temp] + smu_channel_list
    disable_smu(all_smu_channels)
    disconnect_smu([smu_temp, smu_alpha, smu_beta])

    # start to cool down the climate chamber to room temperature
    clim.set_target_temperature(20)
    logging.info("Measurements finished. Starting cooldown procedure.")

    # TODO: analysis of all the Van der Pauw data
    # analyze()

    # if climate chamber reached room temperature switch it off
    clim.go_to_temperature(20, target_accuracy=1, settling_time=5)
    clim.disable()
    logging.info("Cooldown finished. Climate chamber disabled.")


def disable_smu(smu_channel_list):
    """
    Function to disable all outputs of the SMU channels in the channel list
    :param smu_channel_list: list of smu channels that will be disabled
    """
    # put the SMUs in an un-harmful mode
    for device in smu_channel_list:
        device.disable_output()
        device.set_voltage(0)
        device.set_current(0)


def disconnect_smu(smu_list):
    """
    Function to disconnect the smu properly
    :param smu_list: list of SMUs to disconnect
    """
    for device in smu_list:
        device.disconnect()


def go_to_temp(clim, temperature, smua_temp):
    """will cause the climate chamber to go to the specified temperature"""

    logging.info("Wait for the climate chamber to reach " + "{:.2f}".format(temperature) + " °C")

    # command the climate chamber to go to a specified temperature
    # we wait until the climate chamber has approximately the correct temperature
    clim.go_to_temperature(target_temperature=temperature,
                           target_accuracy=TEMPERATURE_ALLOWED_DEVIATION,
                           settling_time=TEMPERATURE_SETTLING_TIME)

    logging.info("Climate chamber reached " + "{:.2f}".format(temperature) + " °C")

    # measure the PT100 and wait for the temperature value to settle
    # now we wait until the sample reached a stable temperature
    wait_for_stabilisation(smua_temp, TEMPERATURE_ALLOWED_DEVIATION, TEMPERATURE_SETTLING_TIME)


def wait_for_stabilisation(smua_temp, max_allowed_deviation, measurement_count):

    deviation = temp_pt100 = 1e10   # some large number to start with
    stable = False
```

```
    readings = []

    logging.info("Waiting for the sample temperature to stabilize.")

    while not stable:
        # take a new reading from the instrument and append it to the readings list
        r_pt100 = smua_temp.measure_resistance()
        temp_pt100 = pt100_r2t(r_pt100)
        readings.append(temp_pt100)

        # if we have already enough readings in the list pop the first (= oldest) element from the list
        # this way the list will always have "count" elements in it.
        if len(readings) > measurement_count:
            readings.pop(0)

            # TODO: Calculate deviation correctly

            # check if the values in the list are stable
            # meaning: is the change in temperature smaller than our allowed deviation
            # if so the temperature is stable and we can exit the loop
            deviation = max(readings) - min(readings)
            if deviation < max_allowed_deviation:
                stable = True

        # log the progress
        logging.info("PT100_temp = " + "{:.2f}".format(temp_pt100) + " °C  |  " +
                     "deviation = " + "{:.2f}".format(deviation) + " °C  |  " +
                     "stability = " + str(stable))

        # sleep for one second before we take the next measurement
        sleep(1)

    logging.info("We have now a stable sample temperature of " + "{:.2f}".format(temp_pt100) + " °C")


def connect_climate_chamber():
    """connect to the climate chamber"""

    # define connection parameters
    ip_address = "129.27.158.42"
    username = "simpacuser"
    password = "u1s2e3r4"

    # connect to the climate chamber
    clim = VT4002(ip_address, username, password)

    return clim


def setup_smu_vdp():
    """connect to the two SMUs used to measure Van der Pauw"""

    # initialize the SMU and connect to it
    smu_alpha = SMU26xx("TCPIP0::129.27.158.189::inst0::INSTR")
    smu_beta = SMU26xx("TCPIP0::129.27.158.41::inst0::INSTR")

    # get the channel object of the SMU
    smu1 = smu_alpha.get_channel(smu_alpha.CHANNEL_A)
    smu2 = smu_alpha.get_channel(smu_alpha.CHANNEL_B)
    smu3 = smu_beta.get_channel(smu_beta.CHANNEL_A)
    smu4 = smu_beta.get_channel(smu_beta.CHANNEL_B)

    # define a list with all the SMUs in it.
    # this enables us to address the smu1 as smu[0]
    # smu_list = [smu1, smu2, smu3, smu4]
    # to test only use smu_alpha
    smu_channel_list = [smu1, smu2, smu3, smu4]

    # reset channels to default settings
    for smu in smu_channel_list:
        smu.reset()

    return [smu_alpha, smu_beta, smu_channel_list]


def setup_smu_temperature():
    """connect to the SMUs (used for PT100 reading)"""

    # initialize the SMU and connect to it
    # we use this smu to measure the the temperature with a PT100 temperature sensor
    smu_temp = SMU26xx("TCPIP0::129.27.158.84::inst0::INSTR")
    smua_temp = smu_temp.get_channel(smu_temp.CHANNEL_A)

    # reset to default settings
    smua_temp.reset()
    # setup the operation mode
    smua_temp.set_mode_current_source()
```

```python
    # set the voltage and current parameters
    smua_temp.set_voltage_range(10)
    smua_temp.set_voltage_limit(10)
    smua_temp.set_voltage(0)

    # we set the measurement current to 1 mA
    smua_temp.set_current_range(1e-3)
    smua_temp.set_current_limit(1e-3)
    smua_temp.set_current(1e-3)

    # set to 4-wire sense mode
    smua_temp.set_sense_4wire()
    # display the resistance on the smu
    smua_temp.display_resistance()
    # set the smu to high accuracy measurement (slower but that doesn't matter)
    smua_temp.set_measurement_speed_hi_accuracy()

    # enable temperature measurement
    smua_temp.enable_output()

    return [smu_temp, smua_temp]


def store_measurement_parameters(filename):
    file = os.path.join(base_path, filename)

    # put all the parameters in a dictionary so we can write it to the json file
    data = {
        'VDP_MEASUREMENT_CURRENT': VDP_MEASUREMENT_CURRENT,
        'VDP_COMPLIANCE_VOLTAGE': VDP_COMPLIANCE_VOLTAGE,
        'VDP_SETTLING_TIME': VDP_SETTLING_TIME,
        'START_TEMPERATURE': START_TEMPERATURE,
        'END_TEMPERATURE': END_TEMPERATURE,
        'TEMPERATURE_STEP_SIZE': TEMPERATURE_STEP_SIZE,
        'TEMPERATURE_SETTLING_TIME': TEMPERATURE_SETTLING_TIME,
        'TEMPERATURE_ALLOWED_DEVIATION': TEMPERATURE_ALLOWED_DEVIATION,
        'SAMPLE_THICKNESS': SAMPLE_THICKNESS,
    }

    # Writing JSON data
    with open(file, 'w') as f:
        json.dump(data, f, indent=4)

    logging.info("Stored the measurement parameters to .\\" + filename)


def configure_logging():

    # get the root logger and set the logging level the user specified
    root_logger = logging.getLogger()
    root_logger.setLevel(LOG_LEVEL)
    logging.getLogger("requests").setLevel(logging.WARNING)

    # define a handler for the log-file
    filename = os.path.join(base_path, "app.log")
    file_handler = logging.FileHandler(filename=filename)
    file_handler.setFormatter(logging.Formatter('%(asctime)s: %(levelname)s: %(message)s'))
    file_handler.setLevel(logging.DEBUG)
    root_logger.addHandler(file_handler)

    # define a log handler that print the output to the stdout
    stream_handler = logging.StreamHandler(sys.stdout)
    stream_handler.setFormatter(logging.Formatter('%(asctime)s: %(levelname)s: %(message)s', datefmt='
      %I:%M:%S'))
    stream_handler.setLevel(logging.INFO)
    root_logger.addHandler(stream_handler)


def create_base_path():
    """creates a folder within the data folder that contains the timestamp"""
    global base_path

    base_directory = "data"
    timestamp = str(strftime("%Y-%m-%d_%H-%M-%S", localtime()))
    base_path = os.path.abspath(os.path.join(base_directory, timestamp))

    if not os.path.exists(base_path):
        os.makedirs(base_path)
    return base_path


if __name__ == '__main__':
    main()
```

sources/vdpResistivityMeasurement.py

```python
"""
```

```
This program measures the resistivity with the Van der Pauw method

Programmed by: Peter Luidolt
Last modified: 2017-02-22
"""

import os.path
import csv
from time import sleep

# import functions to make smu configuration easier
import SMUConfigurations
from libs.UsefulThings import pt100_r2t


def measure_vdp(smu_channel_list,
                test_current,
                compliance_voltage,
                settling_time,
                target_temperature,
                csv_filename,
                logging,
                smua_temp):
    """
    performs a van der Pauw measurement with four SMUs
    :param smu_channel_list: There need to be four SMU channels in that list.
    They must be in the order 1, 2, 3, 4 according to the ASTM Standard
    :param test_current: The current that is applied to the sample
    :param compliance_voltage: The maximum allowed voltage
    :param settling_time: the time between applying a configuration and actual measurement
    :param target_temperature: the temperature at which the measurement takes place
    :param csv_filename: the path to the csv data file the measurements will be written in
    :param logging: the logging object (used to write the log file)
    :param smua_temp: the smu channel that is used to measure the temperature
    :return:
    """

    # check if the csv file already exists and open it
    [file, file_csv, header] = setup_csv_file(csv_filename)

    # dump the header to the logfile so we know what the next debug messages mean
    logging.info(str(header))

    '''make a measurement according to the rotation plan'''

    # define in what order the current is applied
    # we therefore use a list with four entries [A, B, C, D]
    # A ... the SMU that sources the current
    # B ... the SMU that acts as Ground
    # C ... first SMU in high-z mode for voltage measurement
    # D ... second SMU in high-z mode for voltage measurement

    # this rotation plan is according to the resistivity measurement procedure as stated in
    # ASTM F76-08, Standard Test Methods for Measuring Resistivity and Hall Coefficient and
      Determining
    # Hall Mobility in Single-Crystal Semiconductors, ASTM International, West Conshohocken, PA, 2008,
       www.astm.org

    rotation_plan = [[2, 1, 3, 4], [1, 2, 3, 4],
                     [3, 2, 4, 1], [2, 3, 4, 1],
                     [4, 3, 1, 2], [3, 4, 1, 2],
                     [1, 4, 2, 3], [4, 1, 2, 3]]

    # counter
    measurement_number = 0

    # get the start time of the measurement
    # starting_time = datetime.now()

    # make the measurements according to the rotation plan
    for measurement in rotation_plan:

        # increase the measurement counter
        measurement_number += 1

        # the numbers specified in the rotation_plan map to the corresponding SMUs
        # we need to subtract 1 because python starts counting at 0
        source_smu = smu_channel_list[int(measurement[0]) - 1]
        ground_smu = smu_channel_list[int(measurement[1]) - 1]
        high_z_smu_c = smu_channel_list[int(measurement[2]) - 1]
        high_z_smu_d = smu_channel_list[int(measurement[3]) - 1]

        # set all the SMUs into an un-harmful mode
        disable_smu(smu_channel_list)

        # set SMUs in correct modes
        SMUConfigurations.set_smu_to_i_source(source_smu, test_current)
        SMUConfigurations.set_smu_to_ground_connection(ground_smu)
```

```
        SMUConfigurations.set_smu_to_v_measurement(high_z_smu_c)
        SMUConfigurations.set_smu_to_v_measurement(high_z_smu_d)

        # the voltage limit should be set to something ok
        # this highly depends on the device under test (DUT)
        source_smu.set_voltage_range(compliance_voltage)
        source_smu.set_voltage_limit(compliance_voltage)
        source_smu.enable_voltage_autorange()

        # enable high-z and ground smu
        ground_smu.enable_output()
        high_z_smu_c.enable_output()
        high_z_smu_d.enable_output()

        # enable the smu and wait some time till the current is settled
        # print some information for the user
        logging.info("Starting the measurement in configuration: " + str(measurement))

        # enable the source smu
        source_smu.set_current(test_current)
        source_smu.enable_output()

        # wait some time until the voltage is settled
        sleep(settling_time)

        # get the time till start
        # delta_t = datetime.now() - starting_time

        # measure all values that we are interested in
        # we need the source current and the delta voltage between contact C and D
        [source_current, source_voltage] = source_smu.measure_current_and_voltage()
        high_z_voltage_c = high_z_smu_c.measure_voltage()
        high_z_voltage_d = high_z_smu_d.measure_voltage()

        # measure the PT100 temperature sensor
        # we do this so we can check that the temperature isn't changing any more
        r_pt100 = smua_temp.measure_resistance()
        temp_pt100 = pt100_r2t(r_pt100)

        # calculate the delta voltage
        delta_v = high_z_voltage_c - high_z_voltage_d

        # calculate the resistance
        resistance = delta_v / source_current

        # store all the measured values in a dictionary so we can access it later
        vdp_measurement_index = str(measurement[0]) + str(measurement[1]) + str(measurement[2]) + str(
      measurement[3])

        # store values to the csv file
        measurement_data = ([target_temperature,
                             temp_pt100,
                             vdp_measurement_index,
                             source_current,
                             source_voltage,
                             high_z_voltage_c,
                             high_z_voltage_d,
                             delta_v,
                             resistance
                             ])
        file_csv.writerow(measurement_data)

        # ensure that the data is written to the disk immediately
        file.flush()

        # TODO: log measured data
        logging.info(str(measurement_data))

        # set all the SMUs into an un-harmful mode
        disable_smu(smu_channel_list)

    # Close the file properly
    file.close()


def disable_smu(smu_channel_list):
    for smu in smu_channel_list:
        smu.disable_output()
        smu.set_voltage(0)
        smu.set_current(0)


def setup_csv_file(csv_filename):

    # define the headers for the csv file
    header = ['Target Temp (°C)',
              'Actual Temp (°C)',
              'Mode',
```

```
            'I-source (A)',
            'U-source (V)',
            'V-C (V)',
            'V-D (V)',
            'delta_V (V)',
            'Resistance (Ohm)'
            ]

    if os.path.exists(csv_filename):
        # open the file
        f = open(csv_filename, 'a')
        # define the file as csv file
        f_csv = csv.writer(f, lineterminator='\n')

    else:
        # create the file and write the header into it
        f = open(csv_filename, 'w')
        # define the file as csv file
        f_csv = csv.writer(f, lineterminator='\n')

        # write the headers to the new csv file
        f_csv.writerow(header)
        f.flush()

    # return the file and the csv-file object
    return [f, f_csv, header]
```

sources/vdp_measurement.py

```
from libs.KeithleyV13 import _SMUChannel


def set_smu_to_ground_connection(smu):
    """
    Puts the channel into LOW impedance mode.
    This means that current can flow through the smu towards ground.

    Args:
        smu (_SMUChannel): the SMU channel
    """

    # we force the smu to ground potential
    smu.set_mode_voltage_source()
    smu.set_voltage_range(0.2)
    smu.set_voltage(0)

    # let the smu sink / source as much current as needed.
    smu.set_current_range(1)
    smu.set_current_limit(1)
    smu.enable_current_autorange()


def set_smu_to_v_measurement(smu):
    """
    Puts the channel into HIGH impedance mode.
    In this mode the SMU channel can be used to measure voltage towards ground.
    No current should be flowing into / or out of this channel

    Args:
        smu (_SMUChannel): the SMU channel
    """

    # we set the unit to current source (we want to measure the voltage)
    smu.set_mode_current_source()

    # set the smu to the lowest current range and source no current --> the unit will be as High-Z as
      possible
    smu.set_current_range(1e-6)
    smu.set_current_limit(1e-6)
    smu.set_current(0)

    # define a voltage range and limit
    smu.set_voltage_range(200)
    smu.set_voltage_limit(200)
    smu.enable_voltage_autorange()


def set_smu_to_i_source(smu, current_range):
    """
    Puts the channel into current source mode.
    This channel provides us with the measurement current we want to have

    Args:
        smu (_SMUChannel): the SMU channel
        current_range (float): The current the smu channel will source
    """
```

```python
    # we set the unit to current source (we want to measure the voltage)
    smu.set_mode_current_source()

    # set the smu to desired measurement current
    smu.set_current_range(current_range)
    smu.set_current_limit(current_range)
    smu.set_current(0)

    # define a voltage range and limit
    smu.set_voltage_range(200)
    smu.set_voltage_limit(200)
    smu.enable_voltage_autorange()


def set_smu_to_v_source(smu, voltage_range):
    """
    Puts the channel into voltage source mode.
    This channel provides us with the measurement voltage we want to have

    Args:
        smu (_SMUChannel): the SMU channel
        voltage_range (float): the voltage the smu channel will source
    """

    # we set the unit to voltage source (we want to measure the current)
    smu.set_mode_voltage_source()

    # set the smu to the lowest current range and source no current --> the unit will be as High-Z as
      possible
    smu.set_current_range(0.1)
    smu.set_current_limit(0.1)
    smu.enable_current_autorange()

    # set the smu to desired measurement voltage
    smu.set_voltage_range(voltage_range)
    smu.set_voltage_limit(voltage_range)
    smu.set_voltage(0)
```

sources/SMUConfigurations.py

### A.2.3. 2D-Stage Stepper Control code

```c
/*
  2D stage control
  Controls the movement of two carriages with two stepper motors

  created 2016-11-15
  by Peter Luidolt
 */

#include <AccelStepper.h>

// ########################
// constants won't change.

const int ledPin =  13;         // the number of the LED pin
const int xend stopPin = 3;      // the pin the x-Axis end stop switch is connected
const int zend stopPin = 2;      // the pin the x-Axis end stop switch is connected

// define the x stepper
const int xStepperEnablePin = 24;
const int xStepperStepPin = 26;
const int xStepperDirectionPin = 28;
const int xStepperAcceleration = 200; // sets the allowed acceleration of the stepper
const int xStepperMaxSpeed = 1000;    // sets the maximum allowed speed of the stepper
const int xStepperLowerLimit = 0;      // the home position is directly at the left edge
const long xStepperUpperLimit = 7218;   // this equals approx 18 cm to the right
const float xStepperStepsPerCM = 400.5; //defines how many steps are necessary for the carriage to
      move 1 cm

// define the z stepper
const int zStepperEnablePin = A8;
const int zStepperStepPin = 46;
const int zStepperDirectionPin = 48;
const int zStepperAcceleration = 500; // sets the allowed acceleration of the stepper
const int zStepperMaxSpeed = 1000;    // sets the maximum allowed speed of the stepper
const int zStepperLowerLimit = 0;      // the home position is directly at the top edge
const long zStepperUpperLimit = 35000;  // this equals approx 17.6 cm towards the bottom
const float zStepperStepsPerCM = 1988; //defines how many steps are necessary for the carriage to move
       1 cm
```

```
// #######################
// variables that store changing values

int xend stopState = 0;          // variable for reading the x-end stop status
int zend stopState = 0;          // variable for reading the z-end stop status
long XGoToPos = 0;              // variable to store the position the x-servo moves to
long ZGoToPos = 0;              // variable to store the position the z-servo moves to

long SetXPos = 0;               // variable to store the XSET given by the serial command
long SetZPos = 0;               // variable to store the ZSET given by the serial command

// Define a stepper and the pins it will use
AccelStepper xStepper(1,xStepperStepPin,xStepperDirectionPin);   // x stepper
AccelStepper zStepper(1,zStepperStepPin,zStepperDirectionPin);   // z stepper


// #######################
// some functions

// Stop the x stepper as quickly as possible
void xStepperEmergencyStop() {

    xStepper.stop();
    xStepper.setMaxSpeed(0);

    // set a very high acceleration so the stepper can make a quick stop
    xStepper.setAcceleration(200000);

    // actually stops the stepper
    xStepper.runToPosition();

    // move the stepper to the position it already is
    // this should prevent any further movement
    xStepper.move(xStepper.currentPosition());

    // set acceleration and max speed back to the correct value
    xStepper.setAcceleration(xStepperAcceleration);
    xStepper.setMaxSpeed(xStepperMaxSpeed);

    Serial.println("Emergency stop has been hit (x-home limit)");
}

// Stop the z stepper as quickly as possible
void zStepperEmergencyStop() {

    zStepper.stop();
    zStepper.setMaxSpeed(0);

    // set a very high acceleration so the stepper can make a quick stop
    zStepper.setAcceleration(200000);

    // actually stops the stepper
    zStepper.runToPosition();

    // move the stepper to the position it already is
    // this should prevent any further movement
    zStepper.move(zStepper.currentPosition());

    // set acceleration and max speed back to the correct value
    zStepper.setAcceleration(zStepperAcceleration);
    zStepper.setMaxSpeed(zStepperMaxSpeed);

    Serial.println("Emergency stop has been hit (z-home limit)");
}

// moves the x-stepper slowly to the home position
void xFindHome() {

  Serial.println("Slowly moving x-carriage towards home.");

  // xStepper.setMaxSpeed(100);
  // xStepper.setAcceleration(200000);

  // slowly drive to the left
  xStepper.setSpeed(-200);

  // drive left until the end stop is found
  while (digitalRead(xend stopPin) != LOW) {
    xStepper.runSpeed();
  }

  // stop the stepper
  xStepper.stop();
  xStepper.runSpeed();

  // set the end stop position as 0
  xStepper.setCurrentPosition(0);

```

```
  // move a bit away from the end stop (so it is depressed)
  xStepper.moveTo(200);
  xStepper.runToPosition();

  // set the end stop position as 0
  xStepper.setCurrentPosition(0);

  Serial.println("Found x-home!");
  delay(1000);
}

// moves the z-stepper slowly to the home position
void zFindHome() {

  Serial.println("Slowly moving z-carriage towards home.");

  // xStepper.setMaxSpeed(100);
  // xStepper.setAcceleration(200000);

  // slowly drive to the left
  zStepper.setSpeed(-500);

  // drive left until the end stop is found
  while (digitalRead(zend stopPin) != LOW) {
    zStepper.runSpeed();
  }

  // stop the stepper
  zStepper.stop();
  zStepper.runSpeed();

  // set the end stop position as 0
  zStepper.setCurrentPosition(0);

  // move a bit away from the end stop (so it is depressed)
  zStepper.moveTo(600);
  zStepper.runToPosition();

  // set the end stop position as 0
  zStepper.setCurrentPosition(0);

  Serial.println("Found z-home!");
  delay(1000);
}

// ########################
// initialization (runs once when the Arduino boots)

void setup() {

  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  Serial.println("Initializing system ...");

  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);

  // initialize the end stops as input
  pinMode(xend stopPin, INPUT);
  pinMode(zend stopPin, INPUT);

  // initialize the stepper enable pins and enable the steppers
  pinMode(xStepperEnablePin, OUTPUT);
  digitalWrite(xStepperEnablePin, LOW);
  pinMode(zStepperEnablePin, OUTPUT);
  digitalWrite(zStepperEnablePin, LOW);

  // set the maximum speed and the maximum acceleration
  xStepper.setMaxSpeed(xStepperMaxSpeed);
  xStepper.setAcceleration(xStepperAcceleration);
  zStepper.setMaxSpeed(zStepperMaxSpeed);
  zStepper.setAcceleration(zStepperAcceleration);

  // find the x home position by driving slowly to the end stop
  xFindHome();

  // find the z home position by driving slowly to the end stop
  zFindHome();

  // zStepper.moveTo(2000);
  // zStepper.runToPosition();

//  xStepper.moveTo(3000);
//  xStepper.runToPosition();
```

```
  Serial.println("Initialization complete.");
  Serial.println("READY");

}

// ######################
// the loop function runs over and over again forever

void loop() {

  // check if the x-end stop is hit
  if (digitalRead(xend stopPin) == LOW) {
    xStepperEmergencyStop();
    digitalWrite(xStepperEnablePin, HIGH);
    while (1 != 0) {
      // this shouldn't happen so something went terrible wrong
      // for safety we stick here until reset
    }
  }

  // check if the z-end stop is hit
  if (digitalRead(zend stopPin) == LOW) {
    zStepperEmergencyStop();
    digitalWrite(zStepperEnablePin, HIGH);
    while (1 != 0) {
      // this shouldn't happen so something went terrible wrong
      // for safety we stick here until reset
    }
  }


  // check if the is a new command from the serial interface
  if (Serial.available() > 0) {

    // read in the command and store it in a string
    String SerialInStr = Serial.readStringUntil('\r');

    if (SerialInStr == "*IDN?") {
      // returns an identification string
      Serial.println("ARDUINO 2D STAGE CONTROLLER, FIRMWARE v 0.2 (2016-11-15), Peter Luidolt");

    } else if (SerialInStr == "STOP!") {
      // stops the current movement
      xStepper.stop();
      zStepper.stop();

      Serial.println("OK: Stopping all stepper movement.");


    } else if (SerialInStr == "XSET:CENTER") {
      // sets the XSET to the center of the movement range
      // for actual movement the RUN! command needs to be issued

      SetXPos = (xStepperUpperLimit - xStepperLowerLimit)  / 2;

      Serial.println("OK: Set X position to " + String(SetXPos / xStepperStepsPerCM) + " cm / Step-Pos
       = " + String(SetXPos));

    } else if (SerialInStr == "ZSET:CENTER") {
      // sets the ZSET to the center of the movement range
      // for actual movement the RUN! command needs to be issued

      SetZPos = (zStepperUpperLimit - zStepperLowerLimit)  / 2;

      Serial.println("OK: Set Z position to " + String(SetZPos / zStepperStepsPerCM) + " cm / Step-Pos
       = " + String(SetZPos));

    } else if (SerialInStr.startsWith("XSET:")) {
      // x-axis movement

      // Remove the leading XSET: command so that only the number is left and convert this number to a
       float
      SerialInStr.remove(0,5);
      float SerialInNumber = SerialInStr.toFloat();

      // take the number from the serial and calculate how many steps this translates to
      SetXPos = SerialInNumber * xStepperStepsPerCM;

      Serial.println("OK: Set X position to " + String(SerialInNumber) + " cm / Step-Pos = " + String(
     SetXPos));


    } else if (SerialInStr.startsWith("ZSET:")) {
      // z-axis movement

      // Remove the leading ZSET: command so that only the number is left and convert this number to a
       float
```

```
      SerialInStr.remove(0,5);
      float SerialInNumber = SerialInStr.toFloat();

      // take the number from the serial and calculate how many steps this translates to
      SetZPos = SerialInNumber * zStepperStepsPerCM;

      Serial.println("OK: Set Z position to " + String(SerialInNumber) + " cm / Step-Pos = " + String(
     SetZPos));


    } else if (SerialInStr == "RUN!") {
      // actually start the movement if the targets are in range of motion

      // check if the x-value is set outside the range of motion
      if (SetXPos < xStepperLowerLimit || SetXPos > xStepperUpperLimit) {

        Serial.println("ERR: X value out of range --> no movement");

      // check if the z-value is set outside the range of motion
      } else if (SetZPos < zStepperLowerLimit || SetZPos > zStepperUpperLimit) {

        Serial.println("ERR: Z value out of range --> no movement");

      // if the boundaries work out then command the steppers to the new position.
      } else {

        // tell the user that everything is OK and we start moving

        Serial.println("OK: x = " + String(SetXPos / xStepperStepsPerCM) + " cm / z = " + String(
     SetZPos / zStepperStepsPerCM) + " cm" );

        // sets a new target for the x-stepper
        xStepper.moveTo(SetXPos);

        // sets a new target for the z-stepper
        zStepper.moveTo(SetZPos);

      }

    } else if (SerialInStr == "HOME!") {
      // immediately go back to the x=0 and z=0 position
      SetXPos = 0;
      SetZPos = 0;

      Serial.println("OK: x=0 / z=0");

    } else if (SerialInStr == "STATUS?") {
      // return the status of the steppers (moving, stopped, ...)

      if ((xStepper.distanceToGo() == 0) && (zStepper.distanceToGo() == 0)) {
        // Steppers are at the positions they should be
        Serial.println("READY");

      } else {
        // at least one stepper is still moving
        Serial.println("MOVING");
      }

    } else if (SerialInStr == "HELP?") {
      // Print the commands that are available
      Serial.println("List of available commands:");
      Serial.println("HOME!  // Causes the system to go to position 0,0");

      Serial.println("XSET:<cm> // Set the desired absolute x-position");
      Serial.println("ZSET:<cm> // Set the desired absolute z-position");
      Serial.println("XSET:CENTER // Set the x-position to the middle of the range of motion");
      Serial.println("ZSET:CENTER // Set the z-position to the middle of the range of motion");
      Serial.println("RUN!    // Actually performs the movement");

      Serial.println("STOP!   // Stops the movement by deacceleration");

      Serial.println("*IDN?   // Returns the identification");
      Serial.println("HELP?   // Shows the command list");

    } else {
      Serial.println("ERR: UNKNOWN COMMAND");
    }

  } // end of serial command interpretation

  // actually moves the stepper if the current position is different from the set position
  // these lines should be executed as often as possible otherwise the stepper will not move
  xStepper.run();
  zStepper.run();

}
```

sources/2D–Stage–Stepper.ino

## A.2.4. Agilent 3499A switch mainframe library

```
import pyvisa

class Agilent3499A:

    OPERATION_RESET = "*RST"
    OPERATION_CLEAR = "*CLS"
    OPERATION_CLOSE = "ROUTe:CLOSe"  # example: ROUTe:CLOSe (@111)
    OPERATION_OPEN = "ROUTe:OPEN"
    OPERATION_DISPLAY_TEXT = "DIAGnostic:DISPlay"

    def __init__(self, rm=None):

        # variable to store if the debug output was enabled
        self.__debug = False
        self.instrument = None

        # if we have no resource manager then get one
        if rm is None:
            self.rm = pyvisa.ResourceManager()
        else:
            self.rm = rm

    def enable_debug_output(self):
        """Enables the debug output of all communication.The messages will be printed on the console.
        """
        self.__debug = True

    def disable_debug_output(self):
        """Disables the debug output. Nothing will be printed to the console that you haven't
        specified yourself."""
        self.__debug = False

    def connect(self, visa_resource_name):

        # Connect to the device
        self.instrument = self.rm.open_resource(visa_resource_name)

        # define the termination characters as stated in the manual
        self.instrument.read_termination = '\r'
        self.instrument.write_termination = '\r'

        # the instrument handle is returned although the user most likely doesn't need it
        return self.instrument

    def _write(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Write cmd: ' + str(msg))

        # send the command to the instrument
        self.instrument.write(msg)

    def _read(self):
        return self.instrument.read()

    def disconnect(self):
        self.instrument.close()

    def reset(self):
        self._write(self.OPERATION_RESET)
        self._write(self.OPERATION_CLEAR)

    """
    ############################
    Instrument specific functions
    ############################
    """

    def __operate_channel(self, operation, channel):

        # convert the provided channel to a string
        channel_string = str(channel)

        # ensure that we have a three digit channel number
        if len(channel_string) is not 3:
```

```
                raise ValueError("The channel has to have 3 digits in the format YXX. \n"
                                 "Y ... Number of the module\n"
                                 "X ... Number of the channel")

        # construct the message we want to send
        msg = operation + " (@" + channel_string + ")"

        # send it to the instrument
        self._write(msg)

    def close_channel(self, channel):
        self.__operate_channel(self.OPERATION_CLOSE, channel)

    def open_channel(self, channel):
        self.__operate_channel(self.OPERATION_OPEN, channel)

    def display_text(self, text):
        cmd = self.OPERATION_DISPLAY_TEXT + ' "' + str(text) + '"'
        self._write(cmd)


    """
    ############################
    special command to operate the 8x8 switch matrix
    ############################
    """
    @staticmethod
    def calc_real_channel(column, row):

        # calculate the correct module based on the given column and row
        real_column = real_row = switch_module = 0

        # define the four different cases
        if column <= 4 and row <= 4:
            switch_module = 1
            real_row = row
            real_column = column

        elif column > 4 and row <= 4:
            switch_module = 2
            real_row = row
            real_column = column - 4

        elif column <= 4 and row > 4:
            switch_module = 3
            real_row = row - 4
            real_column = column

        elif column > 4 and row > 4:
            switch_module = 4
            real_row = row - 4
            real_column = column - 4

        # the -1 is because Agilent starts to count the channels with number 0
        return str(switch_module) + str(real_row - 1) + str(real_column - 1)

    def close_matrix(self, column, row):
        # calculate the correct module based on the given column and row
        channel = self.calc_real_channel(column, row)
        self.close_channel(channel)
        self.display_text("CLOSED C" + str(column) + ":R" + str(row))

    def open_matrix(self, column, row):
        # calculate the correct module based on the given column and row
        channel = self.calc_real_channel(column, row)
        self.open_channel(channel)
        self.display_text("OPENED C" + str(column) + ":R" + str(row))
```

sources/Agilent3499A.py

### A.2.5. Stanford Research Systems SR830 lock-in Python library

```
import pyvisa

class SR830:
    """library to control / read out the Stanford Research Systems SR830 Lock-In Amplifier"""

    """
    List of device specific commands and parameters based on
```

```
    the programming section (5) of the manual (Starting at page 85)
    """

    # general operations
    OPERATION_IDENTIFY = "*IDN?"
    OPERATION_RESET = "*RST"
    OPERATION_CLEAR = "*CLS"

    # operations concerning communication with the computer
    OPERATION_SEND_RESPONSE_TO_RS232 = "OUTX 0"
    OPERATION_SEND_RESPONSE_TO_GPIB = "OUTX 1"

    # operations / parameters for controlling the oscillator
    OPERATION_SET_TO_INTERNAL_REFERENCE = "FMOD 1"
    OPERATION_SET_TO_EXTERNAL_REFERENCE = "FMOD 0"
    OPERATION_SET_INTERNAL_REFERENCE_FREQUENCY = "FREQ"
    UPPER_FREQ_LIMIT = 102000       # Limit in Hz based on the specifications of the SR830
    LOWER_FREQ_LIMIT = 0.001

    OPERATION_SINE_OUTPUT_LEVEL = "SLVL"
    LOWER_SINE_OUTPUT_LEVEL = 0.004     # Limit in Volts based on the specifications of the SR830
    UPPER_SINE_OUTPUT_LEVEL = 5

    # operations that define the input characteristics
    OPERATION_SET_INPUT_TO_A = "ISRC 0"
    OPERATION_SET_INPUT_TO_A_MINUS_B = "ISRC 1"
    OPERATION_SET_INPUT_SHIELD_TO_FLOATING = "IGND 0"
    OPERATION_SET_INPUT_SHIELD_TO_GROUND = "IGND 1"
    OPERATION_SET_INPUT_COUPLING_AC = "ICPL 0"
    OPERATION_SET_INPUT_COUPLING_DC = "ICPL 1"

    OPERATION_DISABLE_LINE_FILTER = "ILIN 0"
    OPERATION_ENABLE_LINE_FILTER = "ILIN 3"

    # sensitivity commands
    OPERATION_SET_SENSITIVITY = "SENS"
    # Available sensitivity ranges in volts
    SENSITIVITY_RANGES = (2e-9, 5e-9, 10e-9, 20e-9, 50e-9, 100e-9, 200e-9, 500e-9, 1000e-9,
                          2e-6, 5e-6, 10e-6, 20e-6, 50e-6, 100e-6, 200e-6, 500e-6, 1000e-6,
                          2e-3, 5e-3, 10e-3, 20e-3, 50e-3, 100e-3, 200e-3, 500e-3, 1000e-3)

    OPERATION_SET_RESERVE_MODE_HIGH_RESERVE = "RMOD 0"
    OPERATION_SET_RESERVE_MODE_NORMAL = "RMOD 1"
    OPERATION_SET_RESERVE_MODE_LOW_NOISE = "RMOD 2"

    OPERATION_SET_TIME_CONSTANT = "OFLT"
    # Available time constants in seconds
    TIME_CONSTANTS = (10e-6, 30e-6, 100e-6, 300e-6,
                      1e-3, 3e-3, 10e-3, 30e-3, 100e-3, 300e-3,
                      1, 3, 10, 30, 100, 300,
                      1e3, 3e3, 10e3, 30e3)

    OPERATION_LOW_PASS_FILTER_SLOPE = "OFSL"
    # Available filters slopes in dB/oct
    FILTER_SLOPES = (6, 12, 18, 24)

    # display commands
    OPERATION_SET_DISPLAY_CH1_TO_X = "DDEF 1, 0, 0"
    OPERATION_SET_DISPLAY_CH1_TO_R = "DDEF 1, 1, 0"
    OPERATION_SET_DISPLAY_CH2_TO_Y = "DDEF 2, 0, 0"
    OPERATION_SET_DISPLAY_CH2_TO_PHI = "DDEF 2, 1, 0"

    # auto functions
    OPERATION_AUTO_GAIN = "AGAN"
    OPERATION_AUTO_RESERVE = "ARSV"
    OPERATION_AUTO_PHASE = "APHS"
    OPERATION_AUTO_OFFSET_X = "AOFF 1"
    OPERATION_AUTO_OFFSET_Y = "AOFF 2"
    OPERATION_AUTO_OFFSET_R = "AOFF 3"

    # data transfer commands
    READ_X = "OUTP? 1"
    READ_Y = "OUTP? 2"
    READ_R = "OUTP? 3"
    READ_PHI = "OUTP? 4"

    # snap commands read data synchronously (important if time constant is very short)
    READ_SNAP_X_Y_R_PHI = "SNAP? 1, 2, 3, 4"

    """
    General functions to communicate with the device
    """
    def __init__(self, rm=None):

        # variable to store if the debug output was enabled
        self.__debug = False
        self.instrument = None
```

```
        # if we have no resource manager then get one
        if rm is None:
            self.rm = pyvisa.ResourceManager()
        else:
            self.rm = rm

    def enable_debug_output(self):
        """Enables the debug output of all communication.The messages will be printed on the console.
      """
        self.__debug = True

    def disable_debug_output(self):
        """Disables the debug output. Nothing will be printed to the console that you haven't
      specified yourself."""
        self.__debug = False

    def connect(self, visa_resource_name):

        # Connect to the device
        self.instrument = self.rm.open_resource(visa_resource_name)

        # define the termination characters as stated in the manual
        self.instrument.read_termination = '\r'
        self.instrument.write_termination = '\r'

        # clears the resource; if something was in the input buffer it gets lost
        self.instrument.clear()

        # send the appropriate command to respond to RS232 or GIPB based on the initial connection
      method
        if "GPIB" in visa_resource_name:
            # we have a GPIB connection; command the device to also respond to the GPIB interface
            self._write(self.OPERATION_SEND_RESPONSE_TO_GPIB)
        else:
            # send responses to the serial interface
            self._write(self.OPERATION_SEND_RESPONSE_TO_RS232)

        # the instrument handle is returned although the user most likely doesn't need it
        return self.instrument

    def _write(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Write cmd: ' + str(msg))

        # send the command to the instrument
        self.instrument.write(msg)

    def _query(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Query cmd: ' + str(msg))

        # send the command to the instrument
        return self.instrument.query(msg)

    def _read(self):
        return self.instrument.read()

    def disconnect(self):
        self.instrument.close()

    def identify(self):
        return self._query(self.OPERATION_IDENTIFY)

    def reset(self):
        self._write(self.OPERATION_RESET)
        self._write(self.OPERATION_CLEAR)

    """
    Instrument specific functions
    """

    """ Oscillator / reference section """

    def use_external_reference(self):
        self._write(self.OPERATION_SET_TO_EXTERNAL_REFERENCE)

    def use_internal_reference(self):
        self._write(self.OPERATION_SET_TO_INTERNAL_REFERENCE)

    def set_reference_frequency(self, frequency_in_hz):
        if self.LOWER_FREQ_LIMIT <= frequency_in_hz <= self.UPPER_FREQ_LIMIT:
            msg = self.OPERATION_SET_INTERNAL_REFERENCE_FREQUENCY + " " + str(frequency_in_hz)
            self._write(msg)
        else:
            raise ValueError("Frequency must be within " + str(self.LOWER_FREQ_LIMIT) + " Hz to "
                                 + str(self.UPPER_FREQ_LIMIT) + " Hz")
```

```python
    def set_sine_output_level(self, voltage):
        if self.LOWER_SINE_OUTPUT_LEVEL <= voltage <= self.UPPER_SINE_OUTPUT_LEVEL:
            msg = self.OPERATION_SINE_OUTPUT_LEVEL + " " + str(voltage)
            self._write(msg)
        else:
            raise ValueError("Sine output voltage must be within " + str(self.LOWER_SINE_OUTPUT_LEVEL) + " V to "
                             + str(self.UPPER_SINE_OUTPUT_LEVEL) + " V")

    """ Input Mode section """

    def set_input_mode_A(self):
        self._write(self.OPERATION_SET_INPUT_TO_A)

    def set_input_mode_A_minus_B(self):
        self._write(self.OPERATION_SET_INPUT_TO_A_MINUS_B)

    def set_input_shield_to_floating(self):
        self._write(self.OPERATION_SET_INPUT_SHIELD_TO_FLOATING)

    def set_input_shield_to_ground(self):
        self._write(self.OPERATION_SET_INPUT_SHIELD_TO_GROUND)

    def set_input_coupling_ac(self):
        self._write(self.OPERATION_SET_INPUT_COUPLING_AC)

    def set_input_coupling_dc(self):
        self._write(self.OPERATION_SET_INPUT_COUPLING_DC)

    def enable_line_filters(self):
        self._write(self.OPERATION_ENABLE_LINE_FILTER)

    def disable_line_filters(self):
        self._write(self.OPERATION_DISABLE_LINE_FILTER)

    """ sensitivity / time constant section """

    def set_sensitivity(self, sensitivity_in_volt):

        # check the given values for a suitable range and return a value that is certainly available.
        # if the value is larger then the maximum available range, a error is raised
        value = self.find_suitable_range(sensitivity_in_volt, self.SENSITIVITY_RANGES)

        # get the index of the range. This is needed for the command that needs to be sent to the SR830
        range_index = self.SENSITIVITY_RANGES.index(value)

        # construct the command and sent it to the device
        cmd = self.OPERATION_SET_SENSITIVITY + " " + str(range_index)
        self._write(cmd)

    def set_time_constant(self, time_in_seconds):

        # check the given values for a suitable range and return a value that is certainly available.
        # if the value is larger then the maximum available range, a error is raised
        value = self.find_suitable_range(time_in_seconds, self.TIME_CONSTANTS)

        # get the index of the range. This is needed for the command that needs to be sent to the SR830
        range_index = self.TIME_CONSTANTS.index(value)

        # construct the command and sent it to the device
        cmd = self.OPERATION_SET_TIME_CONSTANT + " " + str(range_index)
        self._write(cmd)

    def set_filter_slope(self, filter_in_db):

        # check the given values for a suitable range and return a value that is certainly available.
        # if the value is larger then the maximum available range, a error is raised
        value = self.find_suitable_range(filter_in_db, self.FILTER_SLOPES)

        # get the index of the range. This is needed for the command that needs to be sent to the SR830
        range_index = self.FILTER_SLOPES.index(value)

        # construct the command and sent it to the device
        cmd = self.OPERATION_LOW_PASS_FILTER_SLOPE + " " + str(range_index)
        self._write(cmd)

    """ reserve mode section """

    def set_reserve_high_reserve(self):
        self._write(self.OPERATION_SET_RESERVE_MODE_HIGH_RESERVE)

    def set_reserve_normal(self):
        self._write(self.OPERATION_SET_RESERVE_MODE_NORMAL)
```

```
    def set_reserve_low_noise(self):
        self._write(self.OPERATION_SET_RESERVE_MODE_LOW_NOISE)

    """ display control section (what will be shown on the device display) """

    def display_ch1_x(self):
        self._write(self.OPERATION_SET_DISPLAY_CH1_TO_X)

    def display_ch1_r(self):
        self._write(self.OPERATION_SET_DISPLAY_CH1_TO_R)

    def display_ch2_y(self):
        self._write(self.OPERATION_SET_DISPLAY_CH2_TO_Y)

    def display_ch2_phi(self):
        self._write(self.OPERATION_SET_DISPLAY_CH2_TO_PHI)

    """ auto commands section """

    def auto_gain(self):
        self._write(self.OPERATION_AUTO_GAIN)

    def auto_reserve(self):
        self._write(self.OPERATION_AUTO_RESERVE)

    def auto_phase(self):
        self._write(self.OPERATION_AUTO_PHASE)

    def auto_offset_x(self):
        self._write(self.OPERATION_AUTO_OFFSET_X)

    def auto_offset_y(self):
        self._write(self.OPERATION_AUTO_OFFSET_Y)

    def auto_offset_r(self):
        self._write(self.OPERATION_AUTO_OFFSET_R)

    """ data transfer section section (to read measurement values from the device) """

    def read_x(self):
        return float(self._query(self.READ_X))

    def read_y(self):
        return float(self._query(self.READ_Y))

    def read_r(self):
        return float(self._query(self.READ_R))

    def read_phi(self):
        return float(self._query(self.READ_PHI))

    def read_snap(self):

        # query the values (the values will be read simultaneously and are transmitted together
        response = self._query(self.READ_SNAP_X_Y_R_PHI)
        [x, y, r, phi] = str(response).split(",")

        # convert values to float before returning them
        x = float(x)
        y = float(y)
        r = float(r)
        phi = float(phi)

        return [x, y, r, phi]

    """
    Helper functions
    """

    @staticmethod
    def find_suitable_range(value, value_list):

        # if the value is in the list directly return the given value
        if value in value_list:
            return value

        # if the value is larger then the largest range of the device raise an error.
        # This will maybe prevent the user from overloading the input
        elif value > max(value_list):
            raise ValueError("\n\nThe value " + str(value) + " is larger than the largest available
  range.\n\n" +
                             "Available ranges are:\n" + str(value_list))

        # in other cases just select the smallest possible range the requested value is within
        else:
            # go through the available ranges starting with the smallest and return if we reach a
  suitable range
            for v in sorted(value_list):
```

```
            if v > value:
                return v
```

sources/StanfordResearchSystems.py

## A.2.6. Princeton Applied Research Model 5210 lock-in Python library

```
import pyvisa
from time import sleep


class Model5210:
    """library to control / read out the EG&G Princeton Applied Research Model 5210 Lock-In Amplifier
      """

    """
    List of device specific commands and parameters based on
    the programming section of the manual (chapter 6; starting at page 81)
    """

    # the Model 5210 is not fast enough so we need to wait a bit after the commands we send.
    COMMAND_DELAY = 0.1

    # general operations
    OPERATION_IDENTIFY = "ID; VER"

    # sensitivity commands
    OPERATION_SENSITIVITY = "SEN"
    # Available sensitivity ranges in volts
    SENSITIVITY_RANGES = (100e-9, 300e-9,
                          1e-6, 3e-6, 10e-6, 30e-6, 100e-6, 300e-6,
                          1e-3, 3e-3, 10e-3, 30e-3, 100e-3, 300e-3,
                          1, 3)

    # auto functions
    OPERATION_AUTO_GAIN = "AS"
    OPERATION_AUTO_MEASUREMENT = "ASM"
    OPERATION_AUTO_TUNE_FILTER_FREQUENCY = "ATS"
    OPERATION_AUTO_PHASE = "AQN"
    OPERATION_ABANDON_AUTO_FUNCTION = "AA"

    # line filter functions
    OPERATION_DISABLE_LINE_FILTER = "LF 0"
    OPERATION_ENABLE_LINE_FILTER = "LF 3"

    # main filter options
    OPERATION_SET_FILTER_FLAT = "FLT 0"
    OPERATION_SET_FILTER_NOTCH = "FLT 1"
    OPERATION_SET_FILTER_LP = "FLT 2"
    OPERATION_SET_FILTER_BP = "FLT 3"

    OPERATION_LOW_PASS_FILTER_SLOPE = "XDB"
    # Available filters slopes in dB/oct
    FILTER_SLOPES = (6, 12)

    # operations / parameters for controlling the oscillator
    OPERATION_SET_TO_INTERNAL_REFERENCE = "IE 1"
    OPERATION_SET_TO_EXTERNAL_REFERENCE = "IE 0"

    OPERATION_SINE_OUTPUT_LEVEL = "OA"
    LOWER_SINE_OUTPUT_LEVEL = 0  # Limit in Volts based on the specifications of the Model 5210
    UPPER_SINE_OUTPUT_LEVEL = 2

    # TODO: Oscillator frequency control
    # Manual page 95

    OPERATION_SET_TIME_CONSTANT = "TC"
    # Available time constants in seconds
    TIME_CONSTANTS = (1e-3, 3e-3, 10e-3, 30e-3, 100e-3, 300e-3,
                      1, 3, 10, 30, 100, 300,
                      1e3, 3e3)

    # Dynamic reserve control
    OPERATION_SET_RESERVE_MODE_HIGH_RESERVE = "DR 2"
    OPERATION_SET_RESERVE_MODE_NORMAL = "DR 1"
    OPERATION_SET_RESERVE_MODE_HIGH_STABILITY = "DR 1"

    # display commands
    OPERATION_SET_DISPLAY1_TO_DISP = "D1 5"     # content of display 1 is controlled by display 2
```

```
    OPERATION_SET_DISPLAY2_TO_X_Y_REL = "D2 0"  # Display1: X in %; Display2: Y in %
    OPERATION_SET_DISPLAY2_TO_X_Y_ABS = "D2 0"  # Display1: X in Volt; Display2: Y in Volt
    OPERATION_SET_DISPLAY2_TO_R_PHI = "D2 2"    # Display1: Magnitude; Display2: Phase

    # data transfer commands
    READ_REFERENCE_FREQUENCY = "FRQ"
    READ_X = "X"
    READ_Y = "Y"
    READ_MAG = "MAG"
    READ_PHI = "PHA"
    READ_MAG_PHASE = "MP"

    # status registers
    READ_OVERLOAD_BYTE = "N"

    # There is no RESET option; so we define a State that is "safe" and can be considered similar to a reset.
    # Set sensitivity to 3 V full scale (level 15)
    # Set oscillator output to 0 Volt
    # Set to internal reference
    # Set time constant to 1 second (level 6)
    # Reset the displays to the default view
    OPERATION_RESET = "SEN 15;OA 0;IE 1;TC 6;D1 5;D2 0;"

    """
    General functions to communicate with the device
    """

    def __init__(self, rm=None):

        # variable to store if the debug output was enabled
        self.__debug = False
        self.instrument = None

        # if we have no resource manager then get one
        if rm is None:
            self.rm = pyvisa.ResourceManager()
        else:
            self.rm = rm

    def enable_debug_output(self):
        """Enables the debug output of all communication.The messages will be printed on the console."""
        self.__debug = True

    def disable_debug_output(self):
        """Disables the debug output. Nothing will be printed to the console that you haven't specified yourself."""
        self.__debug = False

    def connect(self, visa_resource_name):

        # Connect to the device
        self.instrument = self.rm.open_resource(visa_resource_name)

        # define the termination characters as stated in the manual
        self.instrument.read_termination = '\r'
        self.instrument.write_termination = '\r'

        # the instrument handle is returned although the user most likely doesn't need it
        return self.instrument

    def _write(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Write cmd: ' + str(msg))

        # send the command to the instrument
        self.instrument.write(msg)
        sleep(self.COMMAND_DELAY)

    def _query(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Query cmd: ' + str(msg))

        # send the command to the instrument
        reading = self.instrument.query(msg)
        sleep(self.COMMAND_DELAY)

        if self.__debug:
            print('Query response: ' + str(reading))

        return reading

    def _read(self):
        # send the command to the instrument
        reading = self.instrument.read()
```

```
        sleep(self.COMMAND_DELAY)

        if self.__debug:
            print('Query response: ' + str(reading))

        return reading

    def disconnect(self):
        self.instrument.close()

    def identify(self):
        return self._query(self.OPERATION_IDENTIFY)

    def reset(self):
        self._write(self.OPERATION_RESET)

    """
    Instrument specific functions
    """

    """ Oscillator / reference section """

    def use_external_reference(self):
        self._write(self.OPERATION_SET_TO_EXTERNAL_REFERENCE)

    def use_internal_reference(self):
        self._write(self.OPERATION_SET_TO_INTERNAL_REFERENCE)

    def set_reference_frequency(self, frequency_in_hz):
        # TODO: Oscillator frequency control
        # Manual page 95
        pass

    def set_sine_output_level(self, voltage):
        if self.LOWER_SINE_OUTPUT_LEVEL <= voltage <= self.UPPER_SINE_OUTPUT_LEVEL:
            msg = self.OPERATION_SINE_OUTPUT_LEVEL + " " + str(voltage)
            self._write(msg)
        else:
            raise ValueError("Sine output voltage must be within " + str(self.LOWER_SINE_OUTPUT_LEVEL)
     + " V to "
                             + str(self.UPPER_SINE_OUTPUT_LEVEL) + " V")

    """ Filter section """

    def enable_line_filters(self):
        self._write(self.OPERATION_ENABLE_LINE_FILTER)

    def disable_line_filters(self):
        self._write(self.OPERATION_DISABLE_LINE_FILTER)

    """ sensitivity / time constant section """

    def set_sensitivity(self, sensitivity_in_volt):

        # check the given values for a suitable range and return a value that is certainly available.
        # if the value is larger then the maximum available range, a error is raised
        value = self.find_suitable_range(sensitivity_in_volt, self.SENSITIVITY_RANGES)

        # get the index of the range. This is needed for the command that needs to be sent to the
      SR830
        range_index = self.SENSITIVITY_RANGES.index(value)

        # construct the command and sent it to the device
        cmd = self.OPERATION_SENSITIVITY + " " + str(range_index)
        self._write(cmd)

    def set_time_constant(self, time_in_seconds):

        # check the given values for a suitable range and return a value that is certainly available.
        # if the value is larger then the maximum available range, a error is raised
        value = self.find_suitable_range(time_in_seconds, self.TIME_CONSTANTS)

        # get the index of the range. This is needed for the command that needs to be sent to the
      SR830
        range_index = self.TIME_CONSTANTS.index(value)

        # construct the command and sent it to the device
        cmd = self.OPERATION_SET_TIME_CONSTANT + " " + str(range_index)
        self._write(cmd)

    def set_filter_slope(self, filter_in_db):

        # check the given values for a suitable range and return a value that is certainly available.
        # if the value is larger then the maximum available range, a error is raised
        value = self.find_suitable_range(filter_in_db, self.FILTER_SLOPES)

        # get the index of the range. This is needed for the command that needs to be sent to the
      SR830
```

```python
        range_index = self.FILTER_SLOPES.index(value)

        # construct the command and sent it to the device
        cmd = self.OPERATION_LOW_PASS_FILTER_SLOPE + " " + str(range_index)
        self._write(cmd)

    """ reserve mode section """

    def set_reserve_high_reserve(self):
        self._write(self.OPERATION_SET_RESERVE_MODE_HIGH_RESERVE)

    def set_reserve_normal(self):
        self._write(self.OPERATION_SET_RESERVE_MODE_NORMAL)

    def set_reserve_high_stability(self):
        self._write(self.OPERATION_SET_RESERVE_MODE_HIGH_STABILITY)

    """ display control section (what will be shown on the device display) """

    def display_x_y_relative(self):
        self._write(self.OPERATION_SET_DISPLAY1_TO_DISP + ";" + self.OPERATION_SET_DISPLAY2_TO_X_Y_REL
  )

    def display_x_y_absolute(self):
        self._write(self.OPERATION_SET_DISPLAY1_TO_DISP + ";" + self.OPERATION_SET_DISPLAY2_TO_X_Y_ABS
  )

    def display_r_phi(self):
        self._write(self.OPERATION_SET_DISPLAY1_TO_DISP + ";" + self.OPERATION_SET_DISPLAY2_TO_R_PHI)

    """ auto commands section """

    def auto_gain(self):
        self._write(self.OPERATION_AUTO_GAIN)

    def auto_phase(self):
        self._write(self.OPERATION_AUTO_PHASE)

    def auto_measurement(self):
        """basically a auto gain and then a auto phase optimisation"""
        self._write(self.OPERATION_AUTO_MEASUREMENT)

    def auto_tune_filter_frequency(self):
        self._write(self.OPERATION_AUTO_TUNE_FILTER_FREQUENCY)

    def stop_auto_function(self):
        self._write(self.OPERATION_ABANDON_AUTO_FUNCTION)

    """ data transfer section section (to read measurement values from the device) """

    def read_reference_frequency(self):
        return self._query(self.READ_REFERENCE_FREQUENCY)

    def read_x(self):
        reading = self._query(self.READ_X)
        return self.calculate_voltage_value(reading)

    def read_y(self):
        reading = self._query(self.READ_Y)
        return self.calculate_voltage_value(reading)

    def read_r(self):
        reading = self._query(self.READ_MAG)
        return self.calculate_voltage_value(reading)

    def read_phi(self):
        """The return from the instrument is in milli-degrees the range +-180000 corresponding to
  +-180°"""
        response = self._query(self.READ_PHI)
        # return a value in degree
        return float(response)/1000

    def read_r_phi(self):
        # query the values (the values will be read simultaneously and are transmitted together
        response = self._query(self.READ_MAG_PHASE)
        [r, phi] = str(response).split(",")

        # convert values to float before returning them
        r = self.calculate_voltage_value(float(r))
        phi = float(phi)/1000

        return [r, phi]

    """
    Helper functions
    """

    def calculate_voltage_value(self, reading):
```

```
        # counter that counts how often we failed to calculate the value
        try_counter = 0

        while try_counter < 5:

            try:
                # get the current sensitivity
                sensitivity_index = int(self._query(self.OPERATION_SENSITIVITY))
                full_scale_sensitivity = self.SENSITIVITY_RANGES[sensitivity_index]

                # the full scale sensitivity is equal to 10000
                voltage = float(reading) * float(full_scale_sensitivity) / 10000
                return voltage

            except ValueError or IndexError:
                # we just try again and keep track how often we tried
                try_counter += 1
                sleep(0.1)
                pass

    @staticmethod
    def find_suitable_range(value, value_list):

        # if the value is in the list directly return the given value
        if value in value_list:
            return value

        # if the value is larger then the largest range of the device raise an error.
        # This will maybe prevent the user from overloading the input
        elif value > max(value_list):
            raise ValueError("\n\nThe value " + str(value) + " is larger than the largest available
 range.\n\n" +
                             "Available ranges are:\n" + str(value_list))

        # in other cases just select the smallest possible range the requested value is within
        else:
            # go through the available ranges starting with the smallest and return if we reach a
 suitable range
            for v in sorted(value_list):
                if v > value:
                    return v
```

sources/PrincetonAppliedResearch.py

## A.2.7. Heinzinger PTN40-125 Power Supply

```
import pyvisa
from time import sleep
from datetime import datetime


class DigitalInterface:

    UNIT_VOLTAGE = "VOLT"
    UNIT_CURRENT = "CURR"

    # the digital interface needs some time to process commands
    # 200 ms proved to be a good value
    COMMAND_DELAY = 0.2

    # It can happen that there is a problem / delay with the network connection
    # in such cases the command is sent again (after some delay).
    # here you can specify how often this will happen until a timeout error is raised
    MAX_RETRIES = 2

    def __init__(self, rm=None):

        # variable to store if the debug output was enabled
        self.__debug = False
        self.instrument = None

        # if we have no resource manager then get one
        if rm is None:
            self.rm = pyvisa.ResourceManager()
        else:
            self.rm = rm

    def connect(self, visa_resource_name):

        # Connect to the device
```

```python
        self.instrument = self.rm.open_resource(visa_resource_name)

        # define the termination characters as stated in the manual
        self.instrument.read_termination = '\00'
        self.instrument.write_termination = '\r'

        # the instrument handle is returned although the user most likely doesn't need it
        return self.instrument

    def disconnect(self):
        self.instrument.close()

    def _query(self, msg, await_return=True):

        # variable to store the data in we will receive
        data = None

        # variables to keep track about success and amount of retries
        retry = 0
        success = False

        # clear anything that is in the queue
        self.instrument.clear()

        # if the command isn't successful the first time we send it again
        # if it fails to often a timeout error will be raised
        while retry < self.MAX_RETRIES and not success:

            try:
                # if the debug output is enabled we dump the msg to the console
                if self.__debug:
                    print('Write cmd: ' + str(msg))

                # send the command to the instrument
                self.instrument.write(msg)
                # the digital interface needs some time to process the command
                sleep(self.COMMAND_DELAY)

                # if we await a return then data is read from the instrument.
                # Otherwise we just set data to True to show the user that the write was successful
                if await_return:
                    data = self.instrument.read()
                    # if the debug output is enabled we dump the response to the console
                    if self.__debug:
                        print('Read: ' + str(data))
                else:
                    data = True

                # if we reach this point the communication was successful
                success = True

            except pyvisa.VisaIOError:
                retry += 1

        # if the command was successful we return the data to the user
        # otherwise we raise a timeout error
        if success:
            return data
        else:
            raise TimeoutError("The command \"" + str(msg) + "\" "
                               + "has been sent " + str(retry+1) + " times "
                               + "but no response has been received.")

    def _write(self, cmd):
        # a write command is just a query without the read
        return self._query(cmd, await_return=False)

    def enable_debug_output(self):
        """Enables the debug output of all communication.The messages will be printed on the console."""
        self.__debug = True

    def disable_debug_output(self):
        """Disables the debug output. Nothing will be printed to the console that you haven't specified yourself."""
        self.__debug = False

    def reset(self):
        self._write("*RST")

    def set_voltage(self, value):
        self._write("VOLT:" + str(value))

    def get_voltage(self):
        return float(self._query("VOLT?"))

    def measure_voltage(self):
        return float(self._query("MEAS:VOLT?"))
```

```
    def set_current(self, value):
        self._write("CURR:" + str(value))

    def get_current(self):
        return float(self._query("CURR?"))

    def measure_current(self):
        return float(self._query("MEAS:CURR?"))

    def identify(self):
        return self._query("IDN?")

    def ramp_voltage(self, target_voltage, slope):
        self._ramp(target_voltage, slope, self.UNIT_VOLTAGE)

    def ramp_current(self, target_current, slope):
        self._ramp(target_current, slope, self.UNIT_CURRENT)

    def _ramp(self, target_value, slope, unit):

        falling = False
        finished = False

        # ensure that slope is positive (no matter what the user entered)
        slope = abs(slope)

        # get the current value (current as up-to-date)
        if unit is self.UNIT_VOLTAGE:
            start_value = self.measure_voltage()
        else:
            start_value = self.measure_current()

        # check if we have a rising or a falling ramp
        if target_value < start_value:
            falling = True
            slope *= -1

        # get the start time and the start value
        starting_time = datetime.now()

        while not finished:

            # info how much time has passed in seconds
            delta_time = (datetime.now() - starting_time).total_seconds()

            # calculate the value that should be set
            value_to_set = (delta_time * slope) + start_value

            # check if we reached our target value
            if falling:
                if value_to_set < target_value:
                    value_to_set = target_value
                    finished = True
            else:
                if value_to_set > target_value:
                    value_to_set = target_value
                    finished = True

            # send the voltage to the device
            if unit is self.UNIT_VOLTAGE:
                self.set_voltage(value_to_set)
            else:
                self.set_current(value_to_set)
```

sources/Heinzinger.py

## A.2.8. MagnetPhysik FH54 Magnetometer

```
import serial                 # Serial communication
import io                     # Input / Output buffer
import re                     # regular expressions
from time import sleep        # sleep command


class FH54:
    """Implements the serial protocol of the Magnet-Physik FH 54 magnetometer"""

    # define Ranges that are available (depends on the probe connected)
    # the Magnet-Physik HS-TGB5-104020 probe has a range from 3mT to 3T
    # RANGE_30uT = 1
```

```python
    # RANGE_300uT = 2
    RANGE_3mT = 3
    RANGE_30mT = 4
    RANGE_300mT = 5
    RANGE_3T = 6
    # RANGE_30T = 7

    ranges_available = {'3mT': RANGE_3mT, '30mT': RANGE_30mT, '300mT': RANGE_300mT, '3T': RANGE_3T}

    def __init__(self):
        # define some variables that are used to store the connection info of the serial interface
        self.__connected = False
        self.__sio = io.TextIOWrapper
        self.__ser = serial.Serial

    def connect(self, port, baud_rate=19200, timeout=1):
        """function used to connect to the magnetometer over the serial interface

        IMPORTANT: This function needs to be called before the other functions are available

        Args:
            port (str): The com port the magnetometer is connected to.
                        under Windows usually something like "COM1"
                        under Linux usually something like "dev/tty1"
            boud_rate (int): The boud rate that is set at the magnetometer
            timeout (int): Timeout for serial commands. You usually don't have to touch this

         Returns:
             bool: The return value. True for success, False otherwise.
        """

        if not self.__connected:
            # connect to the magnetometer
            self.__ser = serial.Serial(port=port, baudrate=baud_rate, timeout=timeout)
            self.__sio = io.TextIOWrapper(io.BufferedRWPair(self.__ser, self.__ser))
            self.__connected = True

            # check if the serial port was opened successfully; if not then open the port
            if not self.__ser.isOpen():
                self.__ser.open()

    def disconnect(self):
        """disconnect the serial port"""
        if self.__connected:
            self.__ser.close()
            self.__connected = False

    def _query(self, cmd):
        """internal function to communicate with the magnetometer"""
        if not self.__connected:
            raise RuntimeError('you need to call connect() first')
        else:
            self.__ser.flushInput()

            # Write the command to the serial port
            self.__sio.write(cmd + '\r')
            self.__sio.flush()  # it is buffering. required to get the data out *now*

            # wait a short time for the multimeter to process the request
            # if we get bytes back then we can read them
            bytes_to_read = 0
            while bytes_to_read == 0:
                sleep(0.1)
                bytes_to_read = self.__ser.inWaiting()

            # read the response and convert it to a string.
            raw_data = self.__ser.read(bytes_to_read)
            return raw_data.decode('utf-8')

    def __query_flag(self, cmd):
        """internal function to query values that have only true or false as answer"""
        raw_data = self._query(str(cmd))

        # filter everything that is not a digit
        value = int(re.sub('[^\d]', '', raw_data))

        if value == 0:
            return False
        else:
            return True

    def read(self):
        """returns the current reading in the unit T (Tesla)"""
        raw_data = self._query('?MEAS')

        if 'FULLS' in raw_data:
            return 'Overflow'
        else:
```

```
            # split the string in value and unit
            data_list = raw_data.split(' ')
            value = float(re.sub('[^\d.-]', '', data_list[0]))
            unit = data_list[1]

            # convert value based on unit reading
            if 'k' in unit:
                value *= 1e3
            elif 'm' in unit:
                value /= 1e3
            elif 'u' in unit:
                value /= 1e6

            return value

    def set_mode_ac(self):
        """sets the device to magnetic AC measurement mode"""
        return self._query('#MODE 1')

    def set_mode_dc(self):
        """sets the device to magnetic DC measurement mode"""
        return self._query('#MODE 0')

    def get_mode(self):
        """returns the current magnetic measurement mode (AC or DC)"""
        raw_data = self._query('?MODE')

        if '0' in raw_data:
            return 'DC'
        else:
            return 'AC'

    @staticmethod
    def get_ranges_available(self):
        """return the available ranges"""
        return self.ranges_available

    def set_range(self, measurement_range):
        """sets the range of the magnetometer
        The parameter range can either be the range-index 1...7 or the word

        examples all commands do the same:
            set_range(3)
            set_range(FH54.RANGE_3mT)
            set_range('3mT')
        """

        # if the range is passed as string
        if 'T' in measurement_range:
            range_id = self.ranges_available[measurement_range]
        else:
            range_id = measurement_range

        # send the command to the device
        raw_data = self._query('#RANGE ' + range_id)
        return raw_data

    def get_range_id(self):
        """returns the range id that corresponds the current range of the magnetometer"""
        raw_data = self._query('?RANGE')

        # filter everything that is not a digit
        value = int(re.sub('[^\d]', '', raw_data))

        return value

    def get_range(self):
        """returns the current range of the magnetometer"""

        current_range_id = self.get_range_id()

        for measurement_range, range_id in self.ranges_available.items():
            if range_id == current_range_id:
                return measurement_range

        return "unknown range with id " + str(current_range_id)

    def set_autorange(self, flag):
        """sets the autoranging mode"""
        if flag:
            return self._query('#AUTO 1')
        else:
            return self._query('#AUTO 0')

    def get_autorange(self):
        """returns True it the magnetometer is in autoranging mode"""
        return self.__query_flag('?AUTO')
```

```
    def zero(self):
        """starts the zero function
        ensure that the zero-field chamber is put over the sensor tip"""
        return self._query('#ZERO 1')

    def get_zero_complete(self):
        """returns True if the zeroing is finished"""
        raw_data = self._query('?ZERO')
        if "OK" in raw_data:
            return True
        else:
            return False

    def set_filter(self, flag):
        """enables or disables the filter"""
        if flag:
            return self._query('#FILTER 1')
        else:
            return self._query('#FILTER 0')

    def get_filter(self):
        """returns if the filter is enabled"""
        return self.__query_flag('?FILTER')

    def set_unit_Tesla(self):
        """sets the displaying unit to Tesla"""
        return self._query('#UNIT 0')

    def set_unit_Gauss(self):
        """sets the displaying unit to Gauss"""
        return self._query('#UNIT 1')

    def set_unit_Ampere_per_meter(self):
        """sets the displaying unit to A/m"""
        return self._query('#UNIT 2')

    def get_unit(self):
        """returns the unit of the device"""
        raw_data = self._query('?UNIT')

        # filter everything that is not a digit
        value = int(re.sub('[^\d]', '', raw_data))

        if value == 0:
            return "T"
        elif value == 1:
            return "G"
        else:
            return "A/m"

    def get_temp(self):
        """returns the current temperature reading"""
        raw_data = self._query('?TEMP')

        # filter everything that is not a digit, the decimal dot or a minus-sign
        value = float(re.sub('[^\d.-]', '', raw_data))
        return value

    def set_temp_off(self):
        """disables the displaying of the temperature"""
        return self._query('#TEMP 0')

    def set_temp_celsius(self):
        """displays the temperature in degree celsius"""
        return self._query('#TEMP 1')

    def set_temp_fahrenheit(self):
        """displays the temperature in degree fahrenheit"""
        return self._query('#TEMP 2')

    def set_limit(self, flag):
        """enables or disables the limit mode"""
        if flag:
            return self._query('#LIMIT 1')
        else:
            return self._query('#LIMIT 0')

    def get_limit(self):
        """returns True if the limit function is enabled"""
        self.__query_flag('?LIMIT')

    def set_upper_limit(self, measurement_range, value, unit=0):
        """sets and enables the upper limit"""
        return self._query('#LIMU ' + str(measurement_range) + ',' + str(value) + ',' + str(unit))

    def set_lower_limit(self, measurement_range, value, unit=0):
        """sets and enables the lower limit"""
        return self._query('#LIML ' + str(measurement_range) + ',' + str(value) + ',' + str(unit))
```

```

    def get_upper_limit(self):
        """returns the current upper limit"""
        raw_data = self._query('?LIMU')
        data_list = raw_data.split(',')
        return data_list

    def get_lower_limit(self):
        """returns the current lower limit"""
        raw_data = self._query('?LIML')
        data_list = raw_data.split(',')
        return data_list

    def set_relative_parameters(self, measurement_range, value, unit=0):
        """sets and enables the relative measurement function"""
        return self._query('#SETREL ' + str(measurement_range) + ',' + str(value) + ',' + str(unit))

    def get_relative_parameters(self):
        """returns the relative measurement parameters"""
        raw_data = self._query('?SETREL')
        data_list = raw_data.split(',')
        return data_list

    def set_relative(self, flag):
        """enables or disables the relative mode"""
        if flag:
            return self._query('#REL 1')
        else:
            return self._query('#REL 0')

    def get_relative(self):
        """returns True if the relative function is enabled"""
        self.__query_flag('?REL')

    def set_peak(self, flag):
        """enables or disables the peak detection mode"""
        if flag:
            return self._query('#PEAK 1')
        else:
            return self._query('#PEAK 0')

    def get_peak(self):
        """returns True if the peak detection mode is enabled

        In DC mode the peak values are displayed in the first line of the display
        and can be accessed by use of the get_meas() function.
        """
        self.__query_flag('?REL')

    def set_max(self, flag):
        """enables or disables the min/max mode"""
        if flag:
            return self._query('#MAX 2')
        else:
            return self._query('#MAX 0')

    def get_max(self, flag):
        """returns True if the magnetometer is in min/max mode"""
        return self.__query_flag('?MAX')

    def read_max(self):
        """read the stored maximum value"""
        raw_data = self._query('?MMAX')

        # split the string in value and unit
        data_list = raw_data.split(' ')
        value = float(re.sub('[^\d.-]', '', data_list[0]))
        unit = data_list[1]

        # convert value based on unit reading
        if 'k' in unit:
            value *= 1e3
        elif 'm' in unit:
            value /= 1e3
        elif 'u' in unit:
            value /= 1e6

        return value

    def read_min(self):
        """read the stored minimum value"""
        raw_data = self._query('?MMIN')

        # split the string in value and unit
        data_list = raw_data.split(' ')
        value = float(re.sub('[^\d.-]', '', data_list[0]))
        unit = data_list[1]
```

```
        # convert value based on unit reading
        if 'k' in unit:
            value *= 1e3
        elif 'm' in unit:
            value /= 1e3
        elif 'u' in unit:
            value /= 1e6

        return value

    def reset_min_max(self):
        """Resets the stored min / max values"""
        return self._query('#RESET')

    def set_local_operation(self):
        """switch to local control"""
        return self._query('#LOCAL')

    def get_local_operation(self):
        """returns True if the unit is in local control mode"""
        return self.__query_flag('?LOCAL')

    def set_number_of_measurements(self, count):
        """defines how many measurements will be returned when started with read_multi_start()
        count = 0 means infinite until read_multi_stop() is executed
        """
        return self._query('#NMEAS ' + str(count))

    def get_number_of_measurements(self):
        """returns the number of measurements that will be executed when started with read_multi_start
    ()
        count = 0 means infinite until read_multi_stop() is executed
        """
        return self._query('?NMEAS')

    def read_multi_start(self):
        """starts the automatic measurement"""
        return self._query('#MULTI 1')

    def read_multi_stop(self):
        """stops the automatic measurement"""
        return self._query('#MULTI 0')

    def read_multi_enabled(self):
        """returns True if the automatic measurement is enabled"""
        return self.__query_flag('?MULTI')

    def set_field_correction(self, flag):
        """enables or disables linearity correction"""
        if flag:
            return self._query('#CFIELD 1')
        else:
            return self._query('#CFIELD 0')

    def get_field_correction(self):
        """returns True if the linearity correction is enabled"""
        self.__query_flag('?CFIELD')

    def set_temp_correction(self, flag):
        """enables or disables temperature correction"""
        if flag:
            return self._query('#CTEMP 1')
        else:
            return self._query('#CTEMP 0')

    def get_temp_correction(self):
        """returns True if the temperature correction is enabled"""
        self.__query_flag('?CTEMP')

    def system_reset(self):
        """executes a system reset of the magnetometer"""
        self.__query_flag('#INIT')
```

sources/MagnetPhysik.py

## A.2.9. Philips PM5193 function generator

```
import pyvisa
from time import sleep

```

```python
class PM5193:
    """library to control / read out the Philips programmable synthesizer / function generator"""

    """
    ##########
    List of device specific commands and parameters based on
    the programming section of the manual (section 1.2.8; starting at page 16)
    ##########
    """

    # the device is not fast enough so we need to wait a bit after the commands we send.
    COMMAND_DELAY = 0.01  # Unit: s

    # general commands
    OPERATION_IDENTIFY = "ID?"

    # frequency commands
    OPERATION_BASE_FREQUENCY = "F"

    # Frequency limits based on the specifications (0.1 mHz to 50 MHz)
    UPPER_FREQUENCY_LIMIT = 50000000
    LOWER_FREQUENCY_LIMIT = 0.0001

    # amplitude commands
    OPERATION_AMPLITUDE_PEAK_PEAK = "LA"
    OPERATION_AMPLITUDE_RMS = "LR"
    OPERATION_AMPLITUDE_DC_OFFSET = "LD"

    # Voltage limits based on the specifications
    # These limits are slightly different for different waveforms.
    LOWER_LIMIT_SINE_VOLTAGE = 0
    UPPER_LIMIT_SINE_VOLTAGE_PEAK_PEAK = 20
    UPPER_LIMIT_SINE_VOLTAGE_RMS = 7

    LOWER_LIMIT_TRIANGULAR_VOLTAGE = 0
    UPPER_LIMIT_TRIANGULAR_VOLTAGE_PEAK_PEAK = 20
    UPPER_LIMIT_TRIANGULAR_VOLTAGE_RMS = 5.7

    LOWER_LIMIT_SQUARE_VOLTAGE = 0
    UPPER_LIMIT_SQUARE_VOLTAGE_PEAK_PEAK = 20
    UPPER_LIMIT_SQUARE_VOLTAGE_RMS = 10

    LOWER_LIMIT_DC_VOLTAGE = -10
    UPPER_LIMIT_DC_VOLTAGE = 10

    # waveform commands
    OPERATION_WAVEFORM_SINE = "WS"
    OPERATION_WAVEFORM_TRIANGULAR = "WS"
    OPERATION_WAVEFORM_SQUARE = "WS"

    # output commands
    OPERATION_AC_OFF = "AC0"
    OPERATION_AC_ON = "AC1"

    # There is no RESET option; so we define a state that is "safe" and can be considered similar to a
      reset.
    # Set DC offset level to 0 Volt
    # Disable AC output
    # Set AC level to 0 Volt p-p
    # Set Waveform to sine
    # Set Frequency to 0 Hz
    OPERATION_RESET = "LD 0; AC0; LA 0; WS; F 0"

    """
    ##########
    General functions to communicate with the device
    ##########
    """

    # if we have no information about the waveform we assume it is sine and set the limits accordingly
    # the values of those variables will change whenever the waveform is changed.
    lower_limit_voltage = LOWER_LIMIT_SINE_VOLTAGE
    upper_limit_voltage_pp = UPPER_LIMIT_SINE_VOLTAGE_PEAK_PEAK
    upper_limit_voltage_rms = UPPER_LIMIT_SINE_VOLTAGE_RMS

    def __init__(self, rm=None):

        # variable to store if the debug output was enabled
        self.__debug = False
        self.instrument = None

        # if we have no resource manager then get one
        if rm is None:
            self.rm = pyvisa.ResourceManager()
        else:
            self.rm = rm

    def enable_debug_output(self):
```

```
        """Enables the debug output of all communication. The messages will be printed on the console.
        """
        self.__debug = True

    def disable_debug_output(self):
        """Disables the debug output. Nothing will be printed to the console that you haven't
        specified yourself."""
        self.__debug = False

    def connect(self, visa_resource_name):

        # Connect to the device
        self.instrument = self.rm.open_resource(visa_resource_name)

        # define the termination characters as stated in the manual
        self.instrument.read_termination = '\r'
        self.instrument.write_termination = '\r'

        # the instrument handle is returned although the user most likely doesn't need it
        return self.instrument

    def _write(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Write cmd: ' + str(msg))

        # send the command to the instrument
        self.instrument.write(msg)
        sleep(self.COMMAND_DELAY)

    def _query(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Query cmd: ' + str(msg))

        # send the command to the instrument
        reading = self.instrument.query(msg)
        sleep(self.COMMAND_DELAY)

        if self.__debug:
            print('Query response: ' + str(reading))

        return reading

    def _read(self):
        # send the command to the instrument
        reading = self.instrument.read()
        sleep(self.COMMAND_DELAY)

        if self.__debug:
            print('Query response: ' + str(reading))

        return reading

    def disconnect(self):
        self.instrument.close()

    def identify(self):
        return self._query(self.OPERATION_IDENTIFY)

    def reset(self):
        self._write(self.OPERATION_RESET)

    """
    ##########
    Instrument specific functions
    ##########
    """

    """ Waveform section """

    def set_waveform_sine(self):
        self.set_voltage_limits(self.OPERATION_WAVEFORM_SINE)
        self._write(self.OPERATION_WAVEFORM_SINE)

    def set_waveform_triangular(self):
        self.set_voltage_limits(self.OPERATION_WAVEFORM_TRIANGULAR)
        self._write(self.OPERATION_WAVEFORM_TRIANGULAR)

    def set_waveform_square(self):
        self.set_voltage_limits(self.OPERATION_WAVEFORM_SQUARE)
        self._write(self.OPERATION_WAVEFORM_SQUARE)

    def set_frequency(self, frequency):
        """Sets the base frequency in Hz"""

        if self.LOWER_FREQUENCY_LIMIT <= frequency <= self.UPPER_FREQUENCY_LIMIT:
            msg = self.OPERATION_BASE_FREQUENCY + " " + str(frequency)
```

```
            self._write(msg)
        else:
            raise ValueError("The output frequency must be within " + str(self.LOWER_FREQUENCY_LIMIT)
 + " Hz to "
                             + str(self.UPPER_FREQUENCY_LIMIT) + " Hz")

    def set_voltage_pp(self, voltage_pp):
        """Sets the Peak-Peak voltage in volts"""

        # check for the correct voltages
        if self.lower_limit_voltage <= voltage_pp <= self.upper_limit_voltage_pp:
            msg = self.OPERATION_AMPLITUDE_PEAK_PEAK + " " + str(voltage_pp)
            self._write(msg)
        else:
            raise ValueError("The output peak to peak voltage must be within " + str(self.
 lower_limit_voltage)
                             + " V to " + str(self.upper_limit_voltage_pp) + " V")

    def set_voltage_rms(self, voltage_rms):
        """Sets the Peak-Peak voltage in volts"""

        # check for the correct voltages
        if self.lower_limit_voltage <= voltage_rms <= self.upper_limit_voltage_rms:
            msg = self.OPERATION_AMPLITUDE_RMS + " " + str(voltage_rms)
            self._write(msg)
        else:
            raise ValueError("The output RMS voltage must be within " + str(self.lower_limit_voltage)
                             + " V to " + str(self.upper_limit_voltage_rms) + " V")

    def set_dc_offset(self, offset_voltage):
        """Sets the DC offset voltage in volts"""

        # check for the correct voltages
        if self.LOWER_LIMIT_DC_VOLTAGE <= offset_voltage <= self.UPPER_LIMIT_DC_VOLTAGE:
            msg = self.OPERATION_AMPLITUDE_DC_OFFSET + " " + str(offset_voltage)
            self._write(msg)
        else:
            raise ValueError("The DC offset voltage must be within " + str(self.LOWER_LIMIT_DC_VOLTAGE
 )
                             + " V to " + str(self.UPPER_LIMIT_DC_VOLTAGE) + " V")

    def disable_ac(self):
        """disables the AC output"""
        self._write(self.OPERATION_AC_OFF)

    def enable_ac(self):
        """enables the AC output"""
        self._write(self.OPERATION_AC_ON)

    """
    ##########
    Helper functions
    ##########
    """

    def set_voltage_limits(self, waveform):
        """set the limits based on the currently selected waveform"""

        if waveform is self.OPERATION_WAVEFORM_SINE:
            self.lower_limit_voltage = self.LOWER_LIMIT_SINE_VOLTAGE
            self.upper_limit_voltage_pp = self.UPPER_LIMIT_SINE_VOLTAGE_PEAK_PEAK
            self.upper_limit_voltage_rms = self.UPPER_LIMIT_SINE_VOLTAGE_RMS

        elif waveform is self.OPERATION_WAVEFORM_TRIANGULAR:
            self.lower_limit_voltage = self.LOWER_LIMIT_TRIANGULAR_VOLTAGE
            self.upper_limit_voltage_pp = self.UPPER_LIMIT_TRIANGULAR_VOLTAGE_PEAK_PEAK
            self.upper_limit_voltage_rms = self.UPPER_LIMIT_TRIANGULAR_VOLTAGE_RMS

        elif waveform is self.OPERATION_WAVEFORM_SQUARE:
            self.lower_limit_voltage = self.LOWER_LIMIT_SQUARE_VOLTAGE
            self.upper_limit_voltage_pp = self.UPPER_LIMIT_SQUARE_VOLTAGE_PEAK_PEAK
            self.upper_limit_voltage_rms = self.UPPER_LIMIT_SQUARE_VOLTAGE_RMS
```

sources/PhilipsPM.py

### A.2.10. Keithley 199 Multimeter

```
import pyvisa


class Keithley199:
```

```
    # define the commands as listed in the Table 3-8 (Device-Dependent Command Summary)
    # on page 3-14 of the Keithley 199 handbook.
    FUNCTION_DC_VOLTS = "F0"
    FUNCTION_AC_VOLTS = "F1"
    FUNCTION_OHMS = "F2"
    FUNCTION_DC_CURRENT = "F3"
    FUNCTION_AC_CURRENT = "F4"
    FUNCTION_ACV_DB = "F5"
    FUNCTION_ACA_DB = "F6"

    RANGE_AUTO = "R0"

    RANGE_300mV = "R1"
    RANGE_3V = "R2"
    RANGE_30V = "R3"
    RANGE_300V = "R4"
    RANGE_30mA = "R1"
    RANGE_3A = "R2"
    RANGE_300Ohm = "R1"
    RANGE_3kOhm = "R2"
    RANGE_30kOhm = "R3"
    RANGE_300kOhm = "R4"
    RANGE_3MOhm = "R5"
    RANGE_30MOhm = "R6"
    RANGE_300MOhm = "R7"

    DEFAULT_CONFIGURATION = "L0"

    OPERATION_EXECUTE = "X"

    DISPLAY = "D"

    def __init__(self, rm=None):

        # variable to store if the debug output was enabled
        self.__debug = False
        self.instrument = None

        # if we have no resource manager then get one
        if rm is None:
            self.rm = pyvisa.ResourceManager()
        else:
            self.rm = rm

    def enable_debug_output(self):
        """Enables the debug output of all communication.The messages will be printed on the console.
        """
        self.__debug = True

    def disable_debug_output(self):
        """Disables the debug output. Nothing will be printed to the console that you haven't
        specified yourself."""
        self.__debug = False

    def connect(self, visa_resource_name):

        # Connect to the device
        self.instrument = self.rm.open_resource(visa_resource_name)

        # define the termination characters as stated in the manual
        self.instrument.read_termination = '\r'
        self.instrument.write_termination = '\r'

        # the instrument handle is returned although the user most likely doesn't need it
        return self.instrument

    def _write(self, msg):
        # if the debug output is enabled we dump the msg to the console
        if self.__debug:
            print('Write cmd: ' + str(msg))

        # send the command to the instrument
        self.instrument.write(msg)

    def _read(self):
        return self.instrument.read()

    def disconnect(self):
        self.instrument.close()

    def reset(self):
        self._write(Keithley199.DEFAULT_CONFIGURATION + Keithley199.OPERATION_EXECUTE)

    """
    ############################
    Instrument specific functions
    ############################
```

```
    """

    def measure(self):

        value = self._read()

        # if we receive a data format that is strange, then command the dmm to use the data format
      without prefix
        # and get the reading once more
        # this will probably only be executed on the first measurement
        if value[0] is not "+" and value[0] is not "-":
            self._write('G1X')
            value = self._read()

        return float(value)

    def set_function_dc_volts(self):
        # puts the device into dc voltage measurement mode
        msg = Keithley199.FUNCTION_DC_VOLTS + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def set_function_ac_volts(self):
        # puts the device into ac voltage measurement mode
        msg = Keithley199.FUNCTION_AC_VOLTS + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def set_function_ohms(self):
        # puts the device into ohms measurement mode
        msg = Keithley199.FUNCTION_OHMS + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def set_function_dc_current(self):
        # puts the device into dc current measurement mode
        msg = Keithley199.FUNCTION_DC_CURRENT + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def set_function_ac_current(self):
        # puts the device into ac current measurement mode
        msg = Keithley199.FUNCTION_AC_CURRENT + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def set_function_ac_volts_db(self):
        # puts the device into ac voltage measurement mode and output the dB
        msg = Keithley199.FUNCTION_ACV_DB + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def set_function_ac_current_db(self):
        # puts the device into ac current measurement mode and output the dB
        msg = Keithley199.FUNCTION_ACA_DB + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def set_range(self, dmm_range):
        msg = dmm_range + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def display_text(self, text):
        """displays a text on the display (max 10 characters)"""

        # we need to substitute spaces with the @ character (the @ is displayed as space on the
      Keithley 199)
        text = text.replace(" ", "@")
        msg = Keithley199.DISPLAY + str(text) + Keithley199.OPERATION_EXECUTE
        self._write(msg)

    def reset_display(self):
        # resets the display to show the current reading again
        msg = Keithley199.DISPLAY + Keithley199.OPERATION_EXECUTE
        self._write(msg)
```

sources/Keithley199.py

### A.2.11. Keithley SourceMeter 2600 series

```
"""
Library to access the basic functionality of the Keithley SourceMeter 2600 series using pyvisa for
      communication.

written by: Peter Luidolt @ TUGraz
last modified: 2016-12-21
"""
```

```python
import pyvisa


# noinspection PyProtectedMember
class _SMUChannel:
    # variables to store the ranges that have been selected
    # we need this information to check if the limit value is valid
    __current_range = 0
    __voltage_range = 0

    def __init__(self, smu_object, smu_channel):
        """
        Implements the functionality for one individual channel of the SMU.

        Args:
            smu_object (SMU26xx): the SMU the channel belongs to
            smu_channel: the channel you want to connect to

        Returns:
            an "channel" object that has methods to control the channel
        """
        # store the parameters in variables that can be accessed from other methods
        self.__smu = smu_object
        self.__channel = smu_channel

    """
    #############
    commands for setting the mode / ranges / limits / levels
    #############
    """

    def identify(self):
        """
        returns a string with model and channel identification
        """
        model = self.__smu.identify_model()

        if self.__channel is SMU26xx.CHANNEL_A:
            channel = "Channel A"
        else:
            channel = "Channel B"

        identification_string = str(model) + " " + str(channel)
        return identification_string

    def reset(self):
        """
        Resets the channel to the default setting of the SMU.
        """
        self.__smu._reset(self.__channel)

    def set_mode_voltage_source(self):
        """
        Sets the channel into voltage source mode.

        In this mode you set the voltage and can measure current, resistance and power.
        """
        self.__smu._set_mode(self.__channel, SMU26xx.VOLTAGE_MODE)

    def set_mode_current_source(self):
        """
        Sets the channel into current source mode.

        In this mode you set the current and can measure voltage, resistance and power.
        """
        self.__smu._set_mode(self.__channel, SMU26xx.CURRENT_MODE)

    def enable_voltage_autorange(self):
        """
        Enables the autorange feature for the voltage source and measurement
        """
        self.__smu._set_autorange(self.__channel, SMU26xx.UNIT_VOLTAGE, SMU26xx.STATE_ON)

    def disable_voltage_autorange(self):
        """
        Disables the autorange feature for the voltage source and measurement
        """
        self.__smu._set_autorange(self.__channel, SMU26xx.UNIT_VOLTAGE, SMU26xx.STATE_OFF)

    def enable_current_autorange(self):
        """
        Enables the autorange feature for the current source and measurement
        """
        self.__smu._set_autorange(self.__channel, SMU26xx.UNIT_CURRENT, SMU26xx.STATE_ON)

    def disable_current_autorange(self):
        """
        Disables the autorange feature for the current source and measurement
```

```
        """
        self.__smu._set_autorange(self.__channel, SMU26xx.UNIT_CURRENT, SMU26xx.STATE_OFF)

    def set_voltage_range(self, value):
        """
        Sets the range for the voltage.

        Args:
            value: set to the maximum expected voltage be sourced or measured

        Examples:
            to set the voltage range to 2 V use:
            >>> self.set_voltage_range(2)

        Note:
            The range is applied to the source function as well as the measurement function.
        """

        # store the requested voltage range; we check it when the limit is set
        self.__voltage_range = value
        self.__smu._set_range(self.__channel, SMU26xx.UNIT_VOLTAGE, value)

    def set_current_range(self, value):
        """
        Sets the range for the current.

        Args:
            value: set to the maximum expected current be sourced or measured

        Examples:
            to set the current range to 100 mA use:
            >>> self.set_voltage_range(0.1)

            you can also use scientific notation: i.e. set the current to 1 uA
            >>> self.set_voltage_range(1e-6)

        Note:
            The range is applied to the source function as well as the measurement function.
        """
        # store the requested current range; we check it when the limit is set
        self.__current_range = value
        self.__smu._set_range(self.__channel, SMU26xx.UNIT_CURRENT, value)

    def set_voltage_limit(self, value):
        """
        Limits the voltage output of the current source.

        Args:
            value: set to the maximum allowed voltage.

        Examples:
            to set the limit to 20 V
            >>> self.set_voltage_limit(20)

        Note:
            If you are in voltage source mode the voltage limit has no effect.

        Raises:
            ValueError: If `value` is bigger then the selected voltage range.
        """

        # check if the limit is within the range
        if value <= self.__voltage_range:
            self.__smu._set_limit(self.__channel, SMU26xx.UNIT_VOLTAGE, value)
        else:
            raise ValueError("The limit is not within the range. Please set the range first")

    def set_current_limit(self, value):
        """
        Limits the current output of the voltage source.

        Args:
            value: set to the maximum allowed current.

        Examples:
            to set the limit to 1 mA (both of the lines below do the same)
            >>> self.set_current_limit(0.001)
            >>> self.set_current_limit(1e-3)

        Note:
            If you are in current source mode the current limit has no effect.

        Raises:
            ValueError: If `value` is bigger then the selected current range.
        """

        # check if the limit is within the range
        if value <= self.__current_range:
```

```
            self.__smu._set_limit(self.__channel, SMU26xx.UNIT_CURRENT, value)
        else:
            raise ValueError("The limit is not within the range. Please set the range first")

    def set_power_limit(self, value):
        """
        Limits the output power.

        Args:
            value: set to the maximum allowed power.
                if you set the `value` to 0 the limit will be disabled

        Examples:
            to set the limit to 1 mW (both of the lines below do the same)
            >>> self.set_power_limit(0.001)
            >>> self.set_power_limit(1e-3)

            to disable the output power limit
            >>> self.set_power_limit(0)
        """
        self.__smu._set_limit(self.__channel, SMU26xx.UNIT_POWER, value)

    def set_voltage(self, value):
        """
        Sets the output level of the voltage source.

        Args:
            value: source voltage level.

        Examples:
            to set the output level to 500 mV
            >>> self.set_voltage(0.5)

        Note:
           If the source is configured as a voltage source and the output is on,
           the new setting is sourced immediately.

           The sign of `level` dictates the polarity of the source.
           Positive values generate positive voltage from the high terminal of the source relative to the low terminal.
           Negative values generate negative voltage from the high terminal of the source relative to the low terminal.
        """
        self.__smu._set_level(self.__channel, SMU26xx.UNIT_VOLTAGE, value)

    def set_current(self, value):
        """
        Sets the output level of the current source.

        Args:
            value: source current level.

        Examples:
            to set the output level to 10 uA
            >>> self.set_current(10e-6)

        Note:
           If the source is configured as a current source and the output is on, the new setting is sourced immediately.

           The sign of `level` dictates the polarity of the source.
           Positive values generate positive current from the high terminal of the source relative to the low terminal.
           Negative values generate negative current from the high terminal of the source relative to the low terminal.
        """
        self.__smu._set_level(self.__channel, SMU26xx.UNIT_CURRENT, value)

    def enable_output(self):
        """
        Sets the source output state to on.

        Examples:
            to enable the output
            >>> self.enable_output()

        Note:
           When the output is switched on, the SMU sources either voltage or current, as set by
           set_mode_voltage_source() or set_mode_current_source()
        """
        self.__smu._set_output_state(self.__channel, SMU26xx.STATE_ON)

    def disable_output(self):
        """
        Sets the source output state to off.

        Examples:
            to disable the output
```

```
            >>> self.disable_output()

        Note:
           When the output is switched off, the SMU goes in to low Z mode (meaning: the output is
  shorted).
           Be careful when using the SMU for measurement of high power devices. The disabling of the
  output could lead
           high current flow.
        """
        self.__smu._set_output_state(self.__channel, SMU26xx.STATE_OFF)

    """
    #############
    commands for setting what measurement will be shown at the display of the SMU channel
    #############
    """

    def display_voltage(self):
        """
        The voltage measurement will be displayed on the SMU.
        """
        self.__smu._set_display(self.__channel, SMU26xx.DISPLAY_VOLTAGE)

    def display_current(self):
        """
        The current measurement will be displayed on the SMU.
        """
        self.__smu._set_display(self.__channel, SMU26xx.DISPLAY_CURRENT)

    def display_resistance(self):
        """
        The calculated resistance will be displayed on the SMU.
        """
        self.__smu._set_display(self.__channel, SMU26xx.DISPLAY_RESISTANCE)

    def display_power(self):
        """
        The calculated power will be displayed on the SMU.
        """
        self.__smu._set_display(self.__channel, SMU26xx.DISPLAY_POWER)

    """
    #############
    commands for setting the sense mode (2-wire or 4-wire)
    #############
    """

    def set_sense_2wire(self):
        """
        Setting the the sense mode to local (2-wire)

        Notes:
            Corresponding LUA command (SMU 2600B reference manual page 2-77)
            smuX.sense = smuX.SENSE_LOCAL
        """
        self.__smu._set_sense_mode(self.__channel, SMU26xx.SENSE_MODE_2_WIRE)

    def set_sense_4wire(self):
        """
        Setting the the sense mode to local (4-wire)

        Notes:
            Corresponding LUA command (SMU 2600B reference manual page 2-77)
            smuX.sense = smuX.SENSE_REMOTE
        """
        self.__smu._set_sense_mode(self.__channel, SMU26xx.SENSE_MODE_4_WIRE)

    """
    #############
    commands for setting the measurement speed / accuracy
    #############
    """

    def set_measurement_speed_fast(self):
        """
        This attribute controls the integration aperture for the analog-to-digital converter (ADC).
        fast corresponds to 0.01 PLC (Power Line Cycles) -> approx. 5000 measurements per second
        Results in: fast performance, but accuracy is reduced
        """
        self.__smu._set_measurement_speed(self.__channel, SMU26xx.SPEED_FAST)

    def set_measurement_speed_med(self):
        """
        This attribute controls the integration aperture for the analog-to-digital converter (ADC).
        fast corresponds to 0.1 PLC (Power Line Cycles) -> approx. 500 measurements per second
        Results in: speed and accuracy are balanced
        """
        self.__smu._set_measurement_speed(self.__channel, SMU26xx.SPEED_MED)
```

```
    def set_measurement_speed_normal(self):
        """
        This attribute controls the integration aperture for the analog-to-digital converter (ADC).
        fast corresponds to 1 PLC (Power Line Cycles) -> approx. 50 measurements per second
        Results in: speed and accuracy are balanced
        """
        self.__smu._set_measurement_speed(self.__channel, SMU26xx.SPEED_NORMAL)

    def set_measurement_speed_hi_accuracy(self):
        """
        This attribute controls the integration aperture for the analog-to-digital converter (ADC).
        fast corresponds to 10 PLC (Power Line Cycles) -> approx. 5 measurements per second
        Results in: high accuracy, but speed is reduced
        """
        self.__smu._set_measurement_speed(self.__channel, SMU26xx.SPEED_HI_ACCURACY)

    """
    #############
    commands for reading values
    #############
    """

    def measure_voltage(self):
        """
        Causes the SMU to trigger a voltage measurement and return a single reading.

        Returns:
            float: the value of the reading in volt
        """
        return self.__smu._measure(self.__channel, SMU26xx.UNIT_VOLTAGE)

    def measure_current(self):
        """
        Causes the SMU to trigger a current measurement and return a single reading.

        Returns:
            float: the value of the reading in ampere
        """
        return self.__smu._measure(self.__channel, SMU26xx.UNIT_CURRENT)

    def measure_resistance(self):
        """
        Causes the SMU to trigger a resistance measurement and return a single reading.

        Returns:
            float: the value of the reading in ohm
        """
        return self.__smu._measure(self.__channel, SMU26xx.UNIT_RESISTANCE)

    def measure_power(self):
        """
        Causes the SMU to trigger a power measurement and return a single reading.

        Returns:
            float: the value of the reading in watt
        """
        return self.__smu._measure(self.__channel, SMU26xx.UNIT_POWER)

    def measure_current_and_voltage(self):
        """
        Causes the SMU to trigger a voltage and current measurement simultaneously.
        Use this function if you need exact time correlation between voltage and current.

        Examples:
            measure current and voltage simultaneously
            >>> [current, voltage] = self.measure_current_and_voltage()

        Returns:
            list: a list of the two measured values.
                current as the first list element
                voltage as the second list element
        """
        return self.__smu._measure(self.__channel, SMU26xx.UNIT_CURRENT_VOLTAGE)

    def measure_voltage_sweep(self, start_value, stop_value, settling_time, points):
        """
        Causes the SMU to make a voltage sweep based on a staircase profile.

        Args:
            start_value: the voltage level from which the sweep will start.
            stop_value: the voltage level at which the sweep will stop.
            settling_time: the time the unit will wait after a voltage step is reached before a
 measurement
                is triggered. If set to 0 the measurement will be done as fast as possible.
            points: the number of steps.

        Note:
```

```
           If you want to measure really fast be sure that you have set the measurement speed accordingly

        Examples:
            perform a voltage sweep from 0 V to 5 V with 500 steps (so 10 mV step size) as fast as possible
            >>> self.set_measurement_speed_fast()
            >>> [current_list, voltage_list] = self.measure_voltage_sweep(0, 5, 0, 500)

        Returns:
            list: the returning list contains itself two lists
                first element is a list of the measured current values
                second element is a list of the voltage source values (not the actual measured voltage)
        """
        return self.__smu._measure_linear_sweep(self.__channel, SMU26xx.UNIT_VOLTAGE,
                                                start_value, stop_value, settling_time, points)

    def measure_current_sweep(self, start_value, stop_value, settling_time, points):
        """
        Causes the SMU to make a current sweep based on a staircase profile.

        Args:
            start_value: the current level from which the sweep will start.
            stop_value: the current level at which the sweep will stop.
            settling_time: the time the unit will wait after a current step is reached before a measurement
                is triggered. If set to 0 the measurement will be done as fast as possible.
            points: the number of steps.

        Note:
           If you want to measure really fast be sure that you have set the measurement speed accordingly

        Examples:
            perform a current sweep from 1 mA to 100 mA with 1000 steps (so 0.1 mA step size)
            and let the device under test 1 second time to settle before taking a measurement
            >>> self.set_measurement_speed_normal()
            >>> [current_list, voltage_list] = self.measure_voltage_sweep(1e-3, 0.1, 1, 1000)

        Returns:
            list: the returning list contains itself two lists
                first element is a list of the current source values (not the actual measured current)
                second element is a list of the measured voltage
        """
        return self.__smu._measure_linear_sweep(self.__channel, SMU26xx.UNIT_CURRENT,
                                                start_value, stop_value, settling_time, points)


class SMU26xx:

    # define strings that are used in the LUA commands
    CHANNEL_A = "a"
    CHANNEL_B = "b"
    # defines an arbitrary word; when used the program tries to access all available channels
    CHANNEL_ALL = "all"

    CURRENT_MODE = "DCAMPS"
    VOLTAGE_MODE = "DCVOLTS"

    DISPLAY_VOLTAGE = 'DCVOLTS'
    DISPLAY_CURRENT = 'DCAMPS'
    DISPLAY_RESISTANCE = 'OHMS'
    DISPLAY_POWER = 'WATTS'

    SENSE_MODE_2_WIRE = 'SENSE_LOCAL'
    SENSE_MODE_4_WIRE = 'SENSE_REMOTE'

    UNIT_VOLTAGE = "v"
    UNIT_CURRENT = "i"
    UNIT_CURRENT_VOLTAGE = "iv"
    UNIT_POWER = "p"
    UNIT_RESISTANCE = "r"

    STATE_ON = "ON"
    STATE_OFF = "OFF"

    SPEED_FAST = 0.01
    SPEED_MED = 0.1
    SPEED_NORMAL = 1
    SPEED_HI_ACCURACY = 10

    # maximum amount of values that can be read from the Keithley buffer without an error from the
    # pyvisa interface. We set it to 1000 values.
    __PYVISA_MAX_BUFFER_REQUEST = 1000

    def __init__(self, visa_resource_name, timeout=1000):
        """
```

```
        Implements the global (channel independent) functionality for the Keithley SMU 2600 series.
        The communication is made through NI-VISA (you need to have this installed)

        Args:
            visa_resource_name: use exactly the VISA-resource-name you see in your NI-MAX

        Returns:
            pyvisa.ResourceManager.open_resource: Object to control the SMU
        """

        # Variables to store the capabilities of the instrument
        self.__voltage_ranges = None
        self.__current_ranges = None
        self.__channel_b_present = None

        # variable to store if the debug output was enabled
        self.__debug = False

        # open the resource manager
        __rm = pyvisa.ResourceManager()

        # Connect to the device
        self.__instrument = __rm.open_resource(visa_resource_name)
        self.__connected = True

        # set the timeout
        self.__instrument.timeout = timeout

        # clear the error queue
        self.__clear_error_queue()

        # clear everything that may is in the buffer
        self.__instrument.clear()

        # find out the ranges of the device and set the limits
        model = self.identify_model()
        self.set_model_limits(model)

    def disconnect(self):
        """
        Disconnect the instrument. After this no further communication is possible.
        """
        if self.__connected:
            self.__instrument.close()
            self.__connected = False

    def get_channel(self, channel):
        """
        Gives you an object with which you can control the individual parameters of a channel.

        Args:
            channel: the channel you want to connect to.
                Use the keywords SMU26xx.CHANNEL_A or SMU26xx.CHANNEL_B

        Returns:
            _SMUChannel: an "channel" object that has methods to control the channel

        Raises:
            ValueError: If the channel is not available.
        """

        # check if the channel b is available. We don't have to check channel a because every smu has
      one
        if channel is SMU26xx.CHANNEL_B and not self.__channel_b_present:
            raise ValueError("No channel B on this model")

        return _SMUChannel(self, channel)

    def enable_debug_output(self):
        """
        Enables the debug output of all communication to the SMU.
        The messages will be printed on the console.
        """
        self.__debug = True

    def disable_debug_output(self):
        """
        Disables the debug output. Nothing will be printed to the console that you haven't specified
      yourself.
        """
        self.__debug = False

    """
    ############
    commands for communicating with the instrument via the pyvisa interface
    ############
    """
```

```python
    def __clear_error_queue(self):
        """
        internal function to clear the error queue of the SMU
        """
        self.write_lua("errorqueue.clear()")

    def __check_error_queue(self):
        """
        requests the error queue from the SMU. If there is an error this function will raise an
        value error containing the message from the SMU.

        Raises:
            ValueError: If there is an error stored at the SMU
        """

        # check if there was an error
        cmd = "errorcode, message = errorqueue.next()\nprint(errorcode, message)"
        response = self.__instrument.query(str(cmd))
        if self.__debug:
            print('Error msg: ' + str(response))
        try:
            [code, message] = response.split('\t', 1)
            if float(code) != 0:
                # if we have an error code something happened and we should raise an error
                raise ValueError('The SMU said: "' + str(message) + '"  /  Keithley-Error-Code: ' +
  str(code))
        except:
            raise ValueError('The SMU said: "' + str(response))

    def write_lua(self, cmd, check_for_errors=True):
        """
        Writes a command to the pyvisa connection. It expects no return message from the SMU

        Args:
            cmd: the TSP command for the SMU
            check_for_errors: by default the error queue of the SMU is checked after every command
  that is send to the
                SMU. In some cases the SMU will not respond to this check and a pyvisa timeout would
  occur. In such
                a case you can disable this check.
        """
        if self.__debug:
            print('Write cmd: ' + str(cmd))
        self.__instrument.write(str(cmd))
        # check if the command executed without any errors
        if check_for_errors:
            self.__check_error_queue()

    def query_lua(self, cmd, check_for_errors=True):
        """
        Queries something from the SMU with TSP syntax.
        basically we just write a TSP command and expect some kind of response from the SMU

        Args:
            cmd: the TSP command for the SMU
            check_for_errors: by default the error queue of the SMU is checked after every command
  that is send to the
                SMU. In some cases the SMU will not respond to this check and a pyvisa timeout would
  occur. In such
                a case you can disable this check.
        """
        if self.__debug:
            print('Query cmd: ' + str(cmd))
        # send the request to the device
        reading = self.__instrument.query(str(cmd)).rstrip('\r\n')
        if self.__debug:
            print('Query answer: ' + str(reading))
        # check if the command executed without any errors
        if check_for_errors:
            self.__check_error_queue()
        return reading

    """
    #############
    commands that gather information of the device and set parameter
    #############
    """

    def identify_model(self):
        """
        Returns the model number of the SMU. Based on this string the model limits are set.

        Returns:
            str: the model number of the SMU
        """
        return self.query_lua('print(localnode.model)')

    def set_model_limits(self, model_number):
```

```
        """
        This function is used to set the model specific differences. This method is called at the initialisation
        process. There is usually no need for you to call this method.

        Args:
            model_number (str): the model number of the SMU.
        """
        if self.__debug:
            print("Model " + str(model_number) + " detected. Setting ranges ...")

        if "2601B" in model_number:
            self.__voltage_ranges = [0.1, 1, 6, 40]
            self.__current_ranges = [1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1, 3]
            self.__channel_b_present = False

        elif "2612A" in model_number:
            self.__voltage_ranges = [0.2, 2, 20, 200]
            self.__current_ranges = [1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1, 1.5]
            self.__channel_b_present = True

        elif "2614B" in model_number:
            self.__voltage_ranges = [0.2, 2, 20, 200]
            self.__current_ranges = [1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1, 1.5]
            self.__channel_b_present = True

        elif "2636A" in model_number:
            self.__voltage_ranges = [0.2, 2, 20, 200]
            self.__current_ranges = [1E-9, 1E-8, 1E-7, 1E-6, 1E-5, 1E-4, 1E-3, 1E-2, 1E-1, 1, 1.5]
            self.__channel_b_present = True
        else:
            raise ValueError("unknown model number")

    def get_available_voltage_ranges(self):
        """
        Returns a list containing the available voltage ranges based on the model limits.

        Returns:
            list: containing the available voltage ranges
        """
        return self.__voltage_ranges

    def get_available_current_ranges(self):
        """
        Returns a list containing the available current ranges based on the model limits.

        Returns:
            list: containing the available current ranges
        """
        return self.__current_ranges

    """
    #############
    commands for measuring values from the two channels simultaneously
    #############
    """

    def measure_voltage(self):
        """
        Causes the SMU to trigger a voltage measurement and return a single reading for both channels (if available).
        Use this function if you need exact time correlation between the voltage of the two channels.

        Examples:
            measure voltage simultaneously on both channels
            >>> [v_chan_a, v_chan_b] = self.measure_voltage()

        Returns:
            list: a list of floats containing the two measured values.
                voltage measurement of channel a as the first list element
                voltage measurement of channel b as the second list element

        Raises:
            ValueError: If the SMU has just one channel
        """
        return self._measure(SMU26xx.CHANNEL_ALL, SMU26xx.UNIT_VOLTAGE)

    def measure_current(self):
        """
        Causes the SMU to trigger a current measurement and return a single reading for both channels (if available).
        Use this function if you need exact time correlation between the current of the two channels.

        Examples:
            measure current simultaneously on both channels
            >>> [i_chan_a, i_chan_b] = self.measure_current()

        Returns:
```

```
            list: a list of floats containing the two measured values.
                current measurement of channel a as the first list element
                current measurement of channel b as the second list element

        Raises:
            ValueError: If the SMU has just one channel
        """
        return self._measure(SMU26xx.CHANNEL_ALL, SMU26xx.UNIT_CURRENT)

    def measure_resistance(self):
        """
        Causes the SMU to trigger a resistance measurement and return a single reading for both channels (if available).
        Use this function if you need exact time correlation between the resistance of the two channels.

        Examples:
            measure resistance simultaneously on both channels
            >>> [r_chan_a, r_chan_b] = self.measure_resistance()

        Returns:
            list: a list of floats containing the two measured values.
                resistance measurement of channel a as the first list element
                resistance measurement of channel b as the second list element

        Raises:
            ValueError: If the SMU has just one channel
        """
        return self._measure(SMU26xx.CHANNEL_ALL, SMU26xx.UNIT_RESISTANCE)

    def measure_power(self):
        """
        Causes the SMU to trigger a power measurement and return a single reading for both channels (if available).
        Use this function if you need exact time correlation between the power of the two channels.

        Examples:
            measure power simultaneously on both channels
            >>> [p_chan_a, p_chan_b] = self.measure_power()

        Returns:
            list: a list of floats containing the two measured values.
                power of channel a as the first list element
                power of channel b as the second list element

        Raises:
            ValueError: If the SMU has just one channel
        """
        return self._measure(SMU26xx.CHANNEL_ALL, SMU26xx.UNIT_POWER)

    def measure_current_and_voltage(self):
        """
        Causes the SMU to trigger a voltage and current measurement simultaneously for both channels (if available).
        Use this function if you need exact time correlation between voltage and current of the two channels.

        Examples:
            measure current and voltage simultaneously on both channels
            >>> [i_chan_a, v_chan_a, i_chan_b, v_chan_b] = self.measure_current_and_voltage()

        Returns:
            list: a list of floats containing the four measured values.
                current of channel a as the first list element
                voltage of channel a as the second list element
                current of channel b as the third list element
                voltage of channel b as the fourth list element

        Raises:
            ValueError: If the SMU has just one channel
        """
        return self._measure(SMU26xx.CHANNEL_ALL, SMU26xx.UNIT_CURRENT_VOLTAGE)

    """
    #############
    commands for setting the parameters of channels
    those should not be accessed directly but through the channel class
    #############
    """

    def _reset(self, channel):
        """restore the default settings"""
        cmd = 'smu' + str(channel) + '.reset()'
        self.write_lua(cmd)

    def _set_display(self, channel, function):
        """defines what measurement will be shown on the display"""
        cmd = 'display.smu' + str(channel) + '.measure.func = display.MEASURE_' + str(function)
```

```
        self.write_lua(cmd)

    def _set_measurement_speed(self, channel, speed):
        """defines how many PLC (Power Line Cycles) a measurement takes"""
        cmd = 'smu' + str(channel) + '.measure.nplc = ' + str(speed)
        self.write_lua(cmd)

    def _set_mode(self, channel, mode):
        cmd = 'smu' + str(channel) + '.source.func = ' + 'smu' + str(channel) + '.OUTPUT_' + str(mode)
        self.write_lua(cmd)

    def _set_sense_mode(self, channel, mode):
        """
        set 2-wire or 4-wire sense mode
        Manual page 2-77

        Notes:
            LUA commands look like this
            smua.sense = smua.SENSE_REMOTE
            smua.sense = smua.SENSE_LOCAL
        """
        cmd = 'smu' + str(channel) + '.sense = ' + 'smu' + str(channel) + '.' + str(mode)
        self.write_lua(cmd)

    def _set_autorange(self, channel, unit, state):
        """enables or disables the autorange feature"""

        # set the source range
        cmd = 'smu' + str(channel) + '.source.autorange' + str(unit) \
              + ' = smu' + str(channel) + '.AUTORANGE_' + str(state)
        self.write_lua(cmd)

        # set the measurement range
        cmd = 'smu' + str(channel) + '.measure.autorange' + str(unit) \
              + ' = smu' + str(channel) + '.AUTORANGE_' + str(state)
        self.write_lua(cmd)

    def _set_range(self, channel, unit, range_value):
        """Set the range to the given value (or to the next suitable range)"""
        range_found = 0

        # select the range you want to compare to based on the given type
        if unit is self.UNIT_CURRENT:
            range_to_check = self.__current_ranges
        elif unit is self.UNIT_VOLTAGE:
            range_to_check = self.__voltage_ranges
        else:
            raise ValueError('Type "' + str(unit) + '" is valid in range setting')

        # find the range that fits the desired value best
        if range_value in range_to_check:
            range_found = range_value
        else:
            # if there is no exact match use the range that is best suitable
            for v in sorted(range_to_check):
                if v > range_value:
                    range_found = v
                    break
            # if none of the ranges above work ... raise an error
            if not range_found:
                raise ValueError("no suitable range found")

        # set the source range
        cmd = 'smu' + str(channel) + '.source.range' + str(unit) + ' = ' + str(range_found)
        self.write_lua(cmd)

        # set the measurement range
        cmd = 'smu' + str(channel) + '.measure.range' + str(unit) + ' = ' + str(range_found)
        self.write_lua(cmd)

    def _set_limit(self, channel, unit, value):
        """command used to set the limits for voltage, current or power"""
        # send the command to the SourceMeter
        cmd = 'smu' + str(channel) + '.source.limit' + str(unit) + ' = ' + str(value)
        self.write_lua(cmd)

    def _set_level(self, channel, unit, value):
        # send the command to the SourceMeter
        cmd = 'smu' + str(channel) + '.source.level' + str(unit) + ' = ' + str(value)
        self.write_lua(cmd)

    def _set_output_state(self, channel, state):
        cmd = 'smu' + str(channel) + '.source.output = smu' + str(channel) + '.OUTPUT_' + str(state)
        self.write_lua(cmd)

    """
    #############
    commands for reading values from the channels
```

```
    those should not be accessed directly but through the channel class
    #############
    """

    def _measure(self, channel, unit):
        """function for getting a single reading of the specified value"""

        # if CHANNEL_ALL is specified this has only an effect on two channel units
        if channel == SMU26xx.CHANNEL_ALL:
            # if channel b is present then modify the LUA command
            if self.__channel_b_present:
                # In case we want to measure voltage and current we get four return parameters
                # so the LUA command has to be different.
                if unit == SMU26xx.UNIT_CURRENT_VOLTAGE:
                    cmd = 'iChA, vChA = smua.measure.' + str(unit) + '()\n' \
                          + 'iChB, vChB = smub.measure.' + str(unit) + '()\n' \
                          + 'print(iChA, vChA, iChB, vChB)'
                else:
                    cmd = 'ChA = smua.measure.' + str(unit) + '()\n' \
                          + 'ChB = smub.measure.' + str(unit) + '()\n' \
                          + 'print(ChA, ChB)'
            else:
                raise ValueError("This device has only ONE channel. "
                                 "Use the measurement function of the channel instead.")
        else:
            cmd = 'print(smu' + str(channel) + '.measure.' + str(unit) + '())'

        reading = self.query_lua(cmd)
        reading = reading.replace("'", "")
        # if we get more than one value out then put it in a list
        out = []
        parts = reading.split("\t")
        if len(parts) > 1:
            for value in parts:
                out.append(float(value))
            return out
        else:
            return float(reading)

    def _measure_linear_sweep(self, channel, unit, start_value, stop_value, settling_time, points):
        """function to sweep voltage or current and measure current resp. voltage"""
        sweep_unit = measure_unit = ''

        if unit is self.UNIT_VOLTAGE:
            sweep_unit = 'V'
            measure_unit = 'I'
        elif unit is self.UNIT_CURRENT:
            sweep_unit = 'I'
            measure_unit = 'V'
        else:
            ValueError('Only possible to sweep Voltage or Current')

        # prepare the buffer
        cmd = 'smu' + str(channel) + '.nvbuffer1.clear()\n' \
              'smu' + str(channel) + '.nvbuffer1.appendmode = 1\n' \
              'smu' + str(channel) + '.nvbuffer1.collectsourcevalues = 1\n' \
              'smu' + str(channel) + '.measure.count = 1'
        self.write_lua(cmd)

        # construct the sweep command based on the given parameters
        # SweepILinMeasureV(smua, 1e-3, 10e-3, 0.1, 10)
        cmd = 'Sweep' + sweep_unit + 'LinMeasure' + measure_unit + '(smu' + str(channel) + ', ' \
              + str(start_value) + ', ' + str(stop_value) + ', ' + str(settling_time) + ', ' + str(
  points) + ')'
        self.write_lua(cmd, check_for_errors=False)

        # wait till the measurement is finished
        # we just try to read some values of the buffer. If we receive an answer we
        # know that the measurement is finished
        answer = None
        cmd = 'print("Are you alive?")'
        while answer is None:
            try:
                # query the values that are stored in the nvbuffer1
                answer = self.query_lua(cmd, check_for_errors=False)
            except pyvisa.VisaIOError:
                # no answer yet ... we just try again
                pass

        # clear any old readings that are in the buffer
        self.__instrument.clear()

        # determine in how many chunks we need to read the buffer and what the start and end values
  are
        quotient = points // self.__PYVISA_MAX_BUFFER_REQUEST
        remainder = points % self.__PYVISA_MAX_BUFFER_REQUEST
        # define the starting values for the buffer read
        buffer_start_values = []
```

```
        buffer_end_values = []
        for i in range(quotient):
            buffer_start_values.append(i * self.__PYVISA_MAX_BUFFER_REQUEST + 1)
            buffer_end_values.append((i+1) * self.__PYVISA_MAX_BUFFER_REQUEST)
        # the last value needs to be set to the amount of data points we have
        if remainder != 0:
            buffer_start_values.append(quotient * self.__PYVISA_MAX_BUFFER_REQUEST + 1)
            buffer_end_values.append(quotient * self.__PYVISA_MAX_BUFFER_REQUEST + remainder)

        # put the readings of the measured data in a list
        measure_values = []
        # read in the buffer and combine the output
        for count in range(len(buffer_start_values)):
            cmd = 'printbuffer(' + str(buffer_start_values[count]) + ', ' + str(buffer_end_values[
      count]) \
                  + ', smu' + str(channel) + '.nvbuffer1.readings)'
            answer = self.query_lua(cmd, check_for_errors=False)
            parts = answer.split(",")
            for value in parts:
                measure_values.append(float(value))
                # clear the visa input buffer
            self.__instrument.clear()

        # put the readings of the source values in a list
        source_values = []
        # read in the buffer and combine the output
        for count in range(len(buffer_start_values)):
            cmd = 'printbuffer(' + str(buffer_start_values[count]) + ', ' + str(buffer_end_values[
      count]) \
                  + ', smu' + str(channel) + '.nvbuffer1.sourcevalues)'
            answer = self.query_lua(cmd, check_for_errors=False)
            parts = answer.split(",")
            for value in parts:
                source_values.append(float(value))
                # clear the visa input buffer
            self.__instrument.clear()

        # always return the current as first parameter
        if unit is self.UNIT_VOLTAGE:
            return [measure_values, source_values]
        else:
            return [source_values, measure_values]
```

sources/KeithleySMU.py

## A.2.12. Hall effect and resistivity measurement

```
"""
This program does a Resistivity measurement during controlled heat up of the cold head
"""

# Import libraries for communication with the devices
from source.libs.Agilent3499A import Agilent3499A
from source.libs.PrincetonAppliedResearch import Model5210
from source.libs.Heinzinger import DigitalInterface
# from source.libs.MagnetPhysik import FH54
from source.libs.PhilipsPM import PM5193
from source.libs.Keithley199 import Keithley199
from source.libs.LakeShore import Model336

# Import some helpful libraries
from source.libs.EngineeringUnitsV2 import ToSI
# from source.libs.UsefulThings import step_list

# Import some other standard libraries
from time import sleep, strftime, localtime
from datetime import datetime
import os  # needed for various operations like creating a directory on the hard drive
import csv  # we use csv to store the results
import json  # We use json to store the parameter file
from pyvisa import VisaIOError  # We need the import to handle errors properly


# Variables used to store the parameters for the measurement; it is declared so early that we can also
       include
# the constants from the next section already
parameters = dict()

# ######
# Define some constants / parameters
# ######
```

```
# just define arbitrary values
RESISTIVITY = 1
HALL = 2

# define what we want to measure during this experiment
MEASUREMENT = HALL

# description that will be put in the parameters file
DESCRIPTION = ["Measures Resistivity or Hall constant during controlled heat up of the cold head"]
parameters.update({"DESCRIPTION": DESCRIPTION})

PROGRAM_FLOW = ["1.) Configure the devices based on the type of measurement",
                "2.) The temperature controller is commanded to heat up the sample",
                "3.) Record resistivity during heat up",
                "4.) If room temperature is reached, end the measurement.",
                "5.) The devices are shut down to a safe state"]
parameters.update({"PROGRAM_FLOW": PROGRAM_FLOW})

# flag to enable / disable the debug output
DEBUG_OUTPUT_ENABLED = False

# value used to estimate the polarity of the measured signal
ACCEPTABLE_PHASE_SHIFT = 30

# parameters for the lock in
TEST_VOLTAGE_RMS = 1
TEST_FREQUENCY = 72
parameters.update({"TEST_VOLTAGE (V_rms)": TEST_VOLTAGE_RMS})
parameters.update({"LOCK_IN_FREQUENCY (Hz)": TEST_FREQUENCY})

# exact value in ohms for the shunt resistor
R_SHUNT = 100.42
parameters.update({"R_SHUNT (Ohm)": R_SHUNT})

# parameters defined by the sample geometry
SAMPLE_CROSS_SECTIONAL_AREA = 240e-12  # Unit: m^2
SAMPLE_RESISTIVITY_TEST_POINT_DISTANCE = 5.5e-3  # Unit: m
SAMPLE_THICKNESS = 80e-9  # Unit: m
parameters.update({"SAMPLE_CROSS_SECTIONAL_AREA (m^2)": SAMPLE_CROSS_SECTIONAL_AREA})
parameters.update({"SAMPLE_LENGTH (m)": SAMPLE_RESISTIVITY_TEST_POINT_DISTANCE})
parameters.update({"SAMPLE_THICKNESS (m)": SAMPLE_THICKNESS})

# measurement parameters
SETTLING_TIME = 20  # Unit: s
parameters.update({"SETTLING_TIME (s)": SETTLING_TIME})

MEASUREMENT_INTERVAL = 1  # Unit: s
parameters.update({"MEASUREMENT_INTERVAL (s)": MEASUREMENT_INTERVAL})

MAGNET_CURRENT = 30  # Unit: A
parameters.update({"MAGNET_CURRENT (A)": MAGNET_CURRENT})

# TODO: either measure the magnetic field or calculate it based on the current through the magnet
# I measured that 30 Amps correspond to approximately 1.04 Tesla; we can work with that for the moment
MAGNETIC_FIELD = 1.04  # Unit: T

# define sensitivities for the lock-in amplifiers
LOCK_IN_SENSITIVITY_RESISTIVITY_MEASUREMENT = 0.3  # Unit: V
LOCK_IN_SENSITIVITY_HALL_MEASUREMENT = 300e-6  # Unit: V
parameters.update({"LOCK_IN_SENSITIVITY_RESISTIVITY_MEASUREMENT (V)":
      LOCK_IN_SENSITIVITY_RESISTIVITY_MEASUREMENT})
parameters.update({"LOCK_IN_SENSITIVITY_HALL_MEASUREMENT (V)": LOCK_IN_SENSITIVITY_HALL_MEASUREMENT})

# define target temperature and heat up characteristics
TARGET_TEMPERATURE = 295  # Unit: K
HEAT_UP_RATE = 10  # Unit: K/min
parameters.update({"TARGET_TEMPERATURE (K)": TARGET_TEMPERATURE})
parameters.update({"HEAT_UP_RATE (K/min)": HEAT_UP_RATE})

# ######
# Main program
# ######

# this flag stores in what state the instruments are. This helps to determine if we need to adjust the
      config
# before we take a reading
current_measurement_setup = None

# variable stores the offset of the hall voltage without magnetic field
v_hall_offset = None


def main():
    """This is the main function where we define the main steps of the program"""

    global current_measurement_setup

```

```
# ------
# initialize all the instruments
# ------
print_delimiter()
print("Initialize measurement devices")

ptn = init_power_supply()
switch = init_switch_matrix()
lia = init_lock_in_model5210()  # lia ... Lock-In Amplifier
fg = init_frequency_generator()
dmm = init_keithley199_dmm()
tc = init_temperature_controller()

# store all the devices in one directory so that it is easier to pass the device handles to other
  functions
devices = {"ptn": ptn,
           "switch": switch,
           "lia": lia,
           "fg": fg,
           "dmm": dmm,
           "tc": tc}

# generates a folder in that we store all the files related with one measurement
data_directory = generate_measurement_directory()

# store the parameters to a json file
write_parameter_file(data_directory)

# ------
# start with the measurement
# ------
print_delimiter()
print("general measurement setup (like frequency generator, closing the circuit to the sample,
  setup csv, ...")

# set the frequency and voltage we want to run the test with.
fg.set_waveform_sine()
fg.set_frequency(TEST_FREQUENCY)
fg.set_voltage_rms(TEST_VOLTAGE_RMS)
fg.enable_ac()

switch.close_matrix(1, 1)  # close the circuit to the sample

# init the csv file to store the resistivity measurement data
f_resistivity_handle = init_csv_file_resistivity(data_directory)

# init the csv file to store the hall measurement data
f_hall_handle = init_csv_file_hall(data_directory)

# get the start time of the measurement
starting_time = datetime.now()

# make one hall measurement without magnetic field to get an offset
measure_hall(devices, starting_time, f_hall_handle)

# ramp up the magnetic field with 1 A/s
ptn.ramp_current(MAGNET_CURRENT, 1)
# sleep(SETTLING_TIME)  # give the system some time to settle

# start the change of the temperature
initiate_temperature_change(devices, TARGET_TEMPERATURE, HEAT_UP_RATE)

# ------
# continuous measurement during heat-up
# ------

# flag to store if the target temperature was reached
target_temperature_reached = False

while not target_temperature_reached:

    measure_hall(devices, starting_time, f_hall_handle)

    # wait for a specified time before taking the next measurement
    sleep(MEASUREMENT_INTERVAL)

    # if we reach room temperature, stop the measurement
    if tc.read_temperature_sensor_D() >= (TARGET_TEMPERATURE - 5):
        target_temperature_reached = True

# end the measurement and bring all devices back to a nice state

# set the sensitivity of the lock in to something un-harming for the input
lia.set_sensitivity(3)

# set the voltage to 0 when you are done with the run
fg.set_voltage_rms(0)
```

```
    switch.open_matrix(1, 1)  # disable power to the sample
    switch.open_matrix(2, 2)  # open the wires for resistivity measurement
    switch.open_matrix(2, 3)  # open the wires for hall measurement

    ptn.ramp_current(0, 1)  # ramp the current (= the magnetic field) back to 0 A

    # ----------------
    # SECTION: tidy up
    # ----------------

    # disable / disconnect / shutdown all devices
    shutdown(ptn, switch, lia, fg, tc)


# ######
# measurement functions
# ######

def measure_hall(devices, starting_time, f_handle):

    # get access to global variables
    global current_measurement_setup
    global v_hall_offset

    # in the f_handle the handle of the csv_file as well as the real file handle is handed over
    # split them up and store them in custom variables so that it is easier to program later on
    f = f_handle[0]
    f_csv = f_handle[1]

    # extract the devices we use and assign them useful names
    temp_controller = devices['tc']  # temperature controller handle
    dmm = devices['dmm']  # digital multi meter
    lia = devices['lia']  # lock in amplifier

    # check if the devices are in hall measurement mode, otherwise set them up properly
    if current_measurement_setup is not HALL:
        setup_hall_measurement(devices)

    # calculate the passed time (delta is in seconds)
    delta = datetime.now() - starting_time
    passed_minutes = float(delta.total_seconds() / 60)

    # take a reading of the temperature values
    temperatures = temp_controller.query_temperatures()

    # measure the current through the sample
    i_sample = dmm.measure()

    # measure the voltage between the test contact for resistivity
    v_hall_end = lia.read_r()
    hall_phase = lia.read_phi()

    # try to estimate the polarity based on the phase shift
    polarity = evaluate_lock_in_polarity(hall_phase)

    if v_hall_offset is None:
        v_hall_offset = v_hall_end

    v_hall = v_hall_end - v_hall_offset
    hall_coefficient = (v_hall * SAMPLE_THICKNESS) / (i_sample * MAGNETIC_FIELD) * 1e6

    # write data to the csv file and flush the buffer (so it is written instantly)
    f_csv.writerow([passed_minutes, delta,
                    temperatures[0], temperatures[1],
                    temperatures[2], temperatures[3],
                    MAGNETIC_FIELD, i_sample,
                    v_hall_offset, v_hall_end, v_hall,
                    hall_phase, polarity, hall_coefficient])
    f.flush()

    print("Time: " + "{:.2f}".format(passed_minutes) + " min  |  "
          + "Temp_D = " + "{:.3f}".format(temperatures[3]) + " K  |  "
          + "I sample = " + ToSI(i_sample, 3) + "A  |  "
          + "V-hall = " + ToSI(v_hall, 3) + "V  |  "
          + "Hall Coefficient = " + str(hall_coefficient * 1e3) + "*10^-3 cm^3/Cl")


def measure_resistivity(devices, starting_time, f_handle):

    # get access to global variables
    global current_measurement_setup

    # in the f_handle the handle of the csv_file as well as the real file handle is handed over
    # split them up and store them in custom variables so that it is easier to program later on
    f = f_handle[0]
    f_csv = f_handle[1]

    # extract the devices we use and assign them useful names
```

```
    temp_controller = devices['tc']  # temperature controller handle
    dmm = devices['dmm']  # digital multi meter
    lia = devices['lia']  # lock in amplifier

    # check if the devices are in resistivity mode, otherwise set them up properly
    if current_measurement_setup is not RESISTIVITY:
        setup_resistivity_measurement(devices)

    # calculate the passed time (delta is in seconds)
    delta = datetime.now() - starting_time
    passed_minutes = float(delta.total_seconds() / 60)

    # take a reading of the temperature values
    temperatures = temp_controller.query_temperatures()

    # measure the current through the sample
    i_sample = dmm.measure()

    # measure the voltage between the test contact for resistivity
    v_resistance = lia.read_r()
    phase_resistance = lia.read_phi()

    # calculate the resistivity based on the sample geometry
    resistance = v_resistance / i_sample
    resistivity = resistance * (SAMPLE_CROSS_SECTIONAL_AREA / SAMPLE_RESISTIVITY_TEST_POINT_DISTANCE)
      * 1e8

    # write data to the csv file and flush the buffer (so it is written instantly)
    f_csv.writerow([passed_minutes, delta,
                    temperatures[0], temperatures[1],
                    temperatures[2], temperatures[3],
                    i_sample, v_resistance, phase_resistance,
                    resistance, resistivity])
    f.flush()

    print("Time: " + "{:.2f}".format(passed_minutes) + " min  |  "
          + "Temp_D = " + "{:.3f}".format(temperatures[3]) + " K  |  "
          + "I sample = " + ToSI(i_sample, 3) + "A  |  "
          + "V-res = " + ToSI(v_resistance, 3) + "V  |  "
          + "Resistivity = " + "{:.3f}".format(resistivity) + " uOhm*cm")


def setup_resistivity_measurement(devices):

    # get access to global variables
    global current_measurement_setup

    # extract the devices we use and assign them useful names
    switch = devices['switch']  # switch matrix handle
    lia = devices['lia']  # lock in amplifier

    # configure the lock in to the highest measurement range (avoids overload)
    lia.set_sensitivity(3)

    # change the switch to the correct setting
    switch.open_matrix(2, 3)  # opens the wires of the hall measurement
    switch.close_matrix(2, 2)  # wires the Lock-In to do resistivity measurement

    # configure the lock in to do resistivity measurement
    lia.set_sensitivity(LOCK_IN_SENSITIVITY_RESISTIVITY_MEASUREMENT)

    # set the flag so that we know which mode we are in
    current_measurement_setup = RESISTIVITY

    # sleep for a bit so that everything can settle
    sleep(SETTLING_TIME)


def setup_hall_measurement(devices):

    # get access to global variables
    global current_measurement_setup

    # extract the devices we use and assign them useful names
    switch = devices['switch']  # switch matrix handle
    lia = devices['lia']  # lock in amplifier

    # configure the lock in to the highest measurement range (avoids overload)
    lia.set_sensitivity(3)

    # change the switch to the correct setting
    switch.open_matrix(2, 2)  # opens the wires of the resistivity measurement
    switch.close_matrix(2, 3)  # wires the Lock-In to do hall measurement

    # configure the lock in to do hall measurement
    lia.set_sensitivity(LOCK_IN_SENSITIVITY_HALL_MEASUREMENT)

    # set the flag so that we know which mode we are in
```

```
    current_measurement_setup = HALL

    # sleep for a bit so that everything can settle
    sleep(SETTLING_TIME)


def initiate_temperature_change(devices, target_temperature, slope):
    """sends the commands to the temperature controller so that we have a controlled heat-up / cool-down"""

    # extract the devices we use and assign them useful names
    tc = devices['tc']  # temperature controller handle

    # disable the setpoint ramp and set the target temperature to the current temperature
    # this way we can run a ramp from the temperature we currently have
    tc.disable_setpoint_ramp()
    tc.set_setpoint(tc.read_temperature_sensor_D())

    # set the slope of the heat up to 5 K/s
    tc.set_setpoint_ramp(slope)
    # set the target temperature to room temperature
    tc.set_setpoint(target_temperature)

    # enable the heater (we can set it to high so it is able to follow the ramp we selected
    tc.set_heater_range_high()


# ######
# Initialisation functions
# ######

def generate_measurement_directory():
    """generates a folder all files are stored in"""

    # Timestamp for the files we store
    timestamp = str(strftime("%Y-%m-%d_%H-%M-%S", localtime()))
    directory = '..\\data\\' + timestamp

    # create the directory for data storage if it doesn't exist
    if not os.path.exists(directory):
        os.makedirs(directory)

    # return the path
    return directory + str("\\")


def init_csv_file_resistivity(directory):
    """setup the csv file we can store the resistivity measurement data in"""

    # open a file and define it as a csv file
    f = open(directory + 'results_resistivity.csv', 'w')
    f_csv = csv.writer(f, delimiter=';', lineterminator='\n')

    # write headers into the file
    f_csv.writerow(['Time (min)', 'Timestamp',
                    'Sensor A (K)', 'Sensor B (K)',
                    'Sensor C (K)', 'Sensor D (K)',
                    'I_sample (A)', 'V_resistivity (V)', 'phase_resistivity (°)',
                    'resistance (Ohm)', 'resistivity (uOhm*cm)'])

    # return the handle to the csv file
    return [f, f_csv]


def init_csv_file_hall(directory):
    """setup the csv file we can store our measurement data in"""

    # open a file and define it as a csv file
    f = open(directory + 'results_hall.csv', 'w')
    f_csv = csv.writer(f, delimiter=';', lineterminator='\n')

    # write headers into the file
    f_csv.writerow(['Time (min)', 'Timestamp',
                    'Sensor A (K)', 'Sensor B (K)',
                    'Sensor C (K)', 'Sensor D (K)',
                    'B field (T)', 'I_sample (A)',
                    'V_hall_offset (V)', 'V_hall_end (V)', 'V_hall (V)',
                    'phase_hall (°)', 'Polarity', 'hall_coefficient (cm^3/Cl)'])

    # return the handle to the csv file
    return [f, f_csv]


def init_temperature_controller():
    """LakeShore 336 temperature controller initialisation"""

    temperature_controller = Model336()
    try:
```

```
        print_delimiter()
        print("Initializing the LakeShore 336 temperature controller  ...")

        # connect to the temperature controller
        temperature_controller.connect("TCPIP0::129.27.158.33::7777::SOCKET")

        # Enable debug output so we see the commands that are sent
        if DEBUG_OUTPUT_ENABLED:
            temperature_controller.enable_debug_output()

        # Reset the device
        # temperature_controller.reset()

        print("Initialisation of the temperature_controller complete.")

        # return the device object so that it can be accessed
        return temperature_controller

    except VisaIOError:
        print("Error initializing the temperature_controller!")
        exit(1)


def init_frequency_generator():
    """Philips PM5193 frequency generator initialisation"""

    fg = PM5193()
    try:
        print_delimiter()
        print("Initializing the Philips frequency generator ...")

        # connect to the switch matrix
        fg.connect("GPIB0::20::INSTR")

        # Enable debug output so we see the commands that are sent over the GPIB interface
        if DEBUG_OUTPUT_ENABLED:
            fg.enable_debug_output()

        # Reset the device
        fg.reset()

        print("Initialisation of the frequency generator complete.")

        # return the device object so that it can be accessed
        return fg

    except VisaIOError:
        print("Error initializing the Philips frequency generator!")
        exit(1)


def init_lock_in_model5210():
    """Princeton Applied Research Model 5210 Lock in Amplifier initialisation"""

    lia = Model5210()  # lia ... Lock-In Amplifier

    try:
        print_delimiter()
        print("Initializing the Princeton Applied Research Model 5210 Lock in Amplifier ...")

        # connect to the Princeton Applied Research Model 5210 Lock in Amplifier
        lia.connect("GPIB0::12::INSTR")

        # Enable debug output so we see the commands that are sent to the instrument
        if DEBUG_OUTPUT_ENABLED:
            lia.enable_debug_output()

        # Reset the device
        lia.reset()

        # configure it for our measurements (we want to measure the shunt resistor)
        lia.use_external_reference()  # use a external reference clock
        # lia.enable_line_filters()  # enable the 2 line filters (50 Hz and 100 Hz)
        lia.disable_line_filters()
        lia.set_time_constant(1)  # set the time constant and the filter
        lia.set_filter_slope(12)
        lia.set_reserve_high_stability()  # set the reserve
        lia.set_sensitivity(3)  # set the sensitivity to 3 V
        lia.display_r_phi()  # set the displays to interesting things

        print("Initialisation of the Princeton Applied Research Model 5210 Lock in Amplifier complete.
      ")

        # return the device object so that it can be accessed
        return lia

    except VisaIOError:
        print("Error initializing the Princeton Applied Research Model 5210 Lock in Amplifier!")
```

```
        exit(1)


def init_keithley199_dmm():
    """Keithley 199 initialisation"""

    dmm = Keithley199()
    try:
        print_delimiter()
        print("Initializing the Keithley 199 dmm...")

        # connect to the device
        dmm.connect("GPIB0::26::INSTR")

        # Enable debug output so we see the commands that are sent over the GPIB interface
        if DEBUG_OUTPUT_ENABLED:
            dmm.enable_debug_output()

        # Reset the device
        dmm.reset()

        dmm.set_function_ac_current()
        dmm.set_range(Keithley199.RANGE_30mA)

        print("Initialisation of the Keithley 199 complete.")

        # return the device object so that it can be accessed
        return dmm

    except VisaIOError:
        print("Error initializing the Keithley 199!")
        exit(1)


def init_switch_matrix():
    """Agilent3499A initialisation"""

    switch = Agilent3499A()
    try:
        print_delimiter()
        print("Initializing the Agilent switch matrix ...")

        # connect to the switch matrix
        switch.connect("GPIB0::2::INSTR")

        # Enable debug output so we see the commands that are sent over the GPIB interface
        if DEBUG_OUTPUT_ENABLED:
            switch.enable_debug_output()

        # Reset the switch matrix (= open all channels)
        switch.reset()

        print("Initialisation of the switch matrix complete.")

        # return the device object so that it can be accessed
        return switch

    except VisaIOError:
        print("Error initializing the switch matrix!")
        exit(1)


def init_power_supply():
    """Magnet power supply initialisation"""

    ptn = DigitalInterface()
    try:
        print_delimiter()
        print("Initializing the magnet power supply ...")

        # connect to the magnet power supply
        ptn.connect("TCPIP0::129.27.158.19::7::SOCKET")

        print("Set initial parameters")
        # initial conditions. We will control the current so the voltage limit has to be high enough
        ptn.set_current(0)
        ptn.set_voltage(100)

        print("Initialisation of the magnet power supply complete.")

        # return the device object so that it can be accessed
        return ptn

    except VisaIOError:
        print("Error initializing the magnet power supply!")
        exit(1)

```

```python
def shutdown(ptn, switch, lia, fg, tc):
    """This function ensures that every instrument is back in an un-harmful state when we finish the measurement
    (or when an error occurs"""

    print_delimiter()
    print("Shutting down all instruments:")

    print("Shutting down magnet power supply to 0 A (Speed: 1 A/s)")
    ptn.ramp_current(0, slope=1)  # gracefully ramp down the magnetic filed
    ptn.set_voltage(0)  # for security set current and voltage to 0
    ptn.set_current(0)
    ptn.disconnect()  # disconnect the power supply

    print("Open all ports on the switch matrix")
    switch.reset()  # open all ports
    switch.disconnect()  # disconnect the switch matrix

    print("Set the Model5210 lock in to an initial state")
    lia.reset()  # set to defaults
    lia.disconnect()

    print("Set the frequency generator to an initial state")
    fg.reset()  # set to defaults
    fg.disconnect()

    print("Turn the heater off and disconnect from the temperature controller")
    tc.set_heater_range_off()
    tc.disconnect()

    print_delimiter()
    print("FINISHED!")


# ######
# Helper functions (not directly related to the experiment; but useful in some way)
# ######

def write_parameter_file(directory):
    """The parameter file is used to store parameters of the setup.
    This should ensure that everyone is able to redo the experiment and reproduce the results."""

    # Writing JSON data
    with open(directory + 'parameters.json', 'w') as f:
        json.dump(parameters, f, indent=4)


def evaluate_lock_in_polarity(phase):
    """check the phase to estimate the polarity"""

    if abs(phase) < ACCEPTABLE_PHASE_SHIFT:
        # if the hall phase is within the acceptable phase shift we can assume that the polarity is positive
        return 1
    elif abs(phase) > (180 - ACCEPTABLE_PHASE_SHIFT):
        # if it is within the phase shift but 180° reversed we can assume that the polarity is negative
        return -1
    else:
        # in other cases it is not wise to decide polarity automatically
        return 0


def print_delimiter():
    """a function that prints a horizontal line. This makes the output in the console nicer"""
    print("---------------------------------------------------------")


# ######
# This is actually the only line that gets executed when you run this file. It checks if you really intended to run it
# (not just import it). If so it executes the main() routine.
# ######

if __name__ == "__main__":
    main()
```

sources/hall–measurement–heatup–rh–v2.py

# Bibliography


[1] M. Ohring. *Reliability and Failure of Electronic Materials and Devices*. Elsevier Science, 1998. ISBN: 9780080516073. URL: https://books.google.at/books?id=gxSyMjosCwcC (cit. on p. 1).

[2] Hiroyuki Kato et al. "Growth and characterization of Ga-doped ZnO layers on a-plane sapphire substrates grown by molecular beam epitaxy." In: *Journal of Crystal Growth* 237-239 (2002), pp. 538–543. ISSN: 00220248. DOI: 10.1016/S0022-0248(01)01972-8 (cit. on p. 1).

[3] Jari Malm et al. "Low-temperature atomic layer deposition of ZnO thin films. Control of crystallinity and orientation." In: *Thin Solid Films* 519 (16 2011), pp. 5319–5322. ISSN: 00406090. DOI: 10.1016/j.tsf.2011.02.024 (cit. on p. 1).

[4] Y. R. Ryu, T. S. Lee, and H. W. White. "Properties of arsenic-doped p-type ZnO grown by hybrid beam deposition." In: *Applied Physics Letters* 83 (1 2003), pp. 87–89. ISSN: 0003-6951. DOI: 10.1063/1.1590423 (cit. on p. 1).

[5] E. H. Hall. "On a New Action of the Magnet on Electric Currents." In: *American Journal of Mathematics* 2 (3 1879), p. 287. ISSN: 00029327. DOI: 10.2307/2369245 (cit. on pp. 1, 4).

[6] Robert Green. *Hall Effect Measurements Essential for Characterizing High Carrier Mobility*. Keithley Instruments, Inc., November 2011 (cit. on pp. 1, 15–17).

[7] Takeshi Ohgaki et al. "Positive Hall coefficients obtained from contact misplacement on evident n-type ZnO films and crystals." In: *Journal of Materials Research* 23 (09 2008), pp. 2293–2295. ISSN: 0884-2914. DOI: 10.1557/jmr.2008.0300 (cit. on p. 1).

[8] P. Wagner and R. Helbig. "Halleffekt und anisotropie der beweglichkeit der elektronen in ZnO." In: *Journal of Physics and Chemistry of Solids* 35 (3 1974), pp. 327–335. ISSN: 00223697. DOI: 10.1016/S0022-3697(74)80026-0 (cit. on p. 1).

[9] P. Drude. "Zur Elektronentheorie der Metalle." In: *Annalen der Physik* 306 (3 1900), pp. 566–613. ISSN: 00033804. DOI: 10.1002/andp.19003060312 (cit. on p. 3).

[10] H. A. Bethe and A. Sommerfeld. *Elektronentheorie der Metalle*. Heidelberger Taschenbücher. Springer-Verlag, 1967. URL: https://books.google.at/books?id=oJ86AAAAMAAJ (cit. on p. 4).

[11] Van der Pauw. "Philips technical review. A Methode of measuring the resistivity and hall coefficient on lamellae of arbitrary shape." In: 20 (8 1958/59), pp. 220–224 (cit. on pp. 7, 13, 14).

[12] Inc. Lake Shore Cryotronics. *7500/9500 Series Hall System User's Manual. Appendix A, Hall effect measurements*. Westerville, Ohio 43082-8888 USA: Lake Shore (cit. on pp. 8, 18).

[13] F01 Committee, ed. *Test Methods for Measuring Resistivity and Hall Coefficient and Determining Hall Mobility in Single-Crystal Semiconductors*. West Conshohocken, PA: ASTM International, 2016. DOI: 10.1520/F0076-08R16E01 (cit. on pp. 8–10, 16, 19, 28).

[14] Wikipedia, ed. *Indiumzinnoxid*. 31.05.2017. URL: https://de.wikipedia.org/w/index.php?oldid=165150667 (visited on 06/02/2017) (cit. on p. 10).

[15] H. Kim et al. "Electrical, optical, and structural properties of indium–tin–oxide thin films for organic light-emitting devices." In: *Journal of Applied Physics* 86 (11 1999), pp. 6451–6461. ISSN: 0021-8979. DOI: 10.1063/1.371708 (cit. on pp. 10, 11, 59).

[16] Toshiro Maruyama and Kunihiro Fukui. "Indium tin oxide thin films prepared by chemical vapour deposition." In: *Thin Solid Films* 203 (2 1991), pp. 297–302. ISSN: 00406090. DOI: 10.1016/0040-6090(91)90137-M (cit. on p. 10).

[17] Wen-Fa, Wu and Bi-Shiou Chiou. "Effect of oxygen concentration in the sputtering ambient on the microstructure, electrical and optical properties of radio-frequency magnetron-sputtered indium tin oxide films." In: *Semiconductor Science and Technology* 11 (2 1996), p. 196. ISSN: 0268-1242 (cit. on p. 10).

[18] I. A. Rauf. "Structure and properties of tin-doped indium oxide thin films prepared by reactive electron-beam evaporation with a zone-confining arrangement." In: *Journal of Applied Physics* 79 (8 1996), p. 4057. ISSN: 0021-8979. DOI: 10.1063/1.361882 (cit. on p. 10).

[19] Keithley Instruments, Inc. *Low Level Measurement Handbook. Precision DC Current, Voltage, and Resistance Measurements*. Version 7th Edition. 2013. URL: http://www.tek.com/document/primer/low-level-measurements-handbook-precision-dc-current-voltage-and-resistance-measurem# (cit. on p. 22).

[20] Zurich Instruments. *Principles of lock-in detection and the state of the art (white paper)*. November 2016 (cit. on pp. 23, 24).

[21] Standford Research Systems. *Model SR830 Manual. DSP Lock-In Amplifier*. Version Revision 2.5. October 2011 (cit. on pp. 24, 97).

[22] Keithley Instruments, Inc. *Series 2600B System SourceMeter Instrument Reference Manual*. 2600BS-901-01 Rev. A. Cleveland, Ohio, U.S.A., September 2012 (cit. on pp. 46, 91).

[23] Ocal Tuna et al. "High quality ITO thin films grown by dc and RF sputtering without oxygen." In: *Journal of Physics D: Applied Physics* 43 (5 2010), p. 055402. ISSN: 0022-3727. DOI: 10.1088/0022-3727/43/5/055402 (cit. on p. 56).

[24] Michael Stowell et al. "RF-superimposed DC and pulsed DC sputtering for deposition of transparent conductive oxides." In: *Thin Solid Films* 515 (19 2007), pp. 7654–7657. ISSN: 00406090. DOI: 10.1016/j.tsf.2006.11.166 (cit. on p. 56).

[25] Naoki Nishimoto et al. "Effect of temperature on the electrical properties of ITO in a TiO 2 /ITO film." In: *physica status solidi (a)* 210 (3 2013), pp. 589–593. ISSN: 18626300. DOI: 10.1002/pssa.201228325 (cit. on p. 56).

[26] L. H.W. van Beveren et al. "Indium Tin Oxide film characterization using the classical Hall Effect." In: *2014 CONFERENCE ON OPTOELECTRONIC AND MICROELECTRONIC MATERIALS AND DEVICES (COMMAD 2014)* (2014) (cit. on pp. 57, 60).

[27] Wikipedia, ed. *Indium tin oxide - Wikipedia*. 9.06.2017. URL: https://en.wikipedia.org/w/index.php?oldid=781283039 (visited on 06/19/2017) (cit. on p. 59).

[28] John Pern. *Stability Issues of Transparent Conducting Oxides (TCOs) for Thin-Film Photovoltaics*. In collab. with APP International PV Reliability Workshop. Golden, Colorado, USA: National Renewable Energy Laboratory, USA, Dec. 1, 2008 (cit. on p. 60).

[29] Heinzinger electronic GmbH. *Operating instructions. PTN 125 - 40 / 72IP*. Rosenheim, Germany, August 2012 (cit. on p. 66).

[30] Inc. Advanced Research Systems. *Operation Manual Expanders. Models DE-202 and DE-204*. Rev 4. Macungie, PA 18062, U.S.A., November 2012 (cit. on pp. 69, 75).

[31] Inc. Advanced Research Systems. *Technical Manual Model ARS-4HW. Water-cooled helium compressor*. Rev 3. Macungie, PA 18062, U.S.A., November 2012 (cit. on p. 69).

[32] Inc. Lake Shore Cryotronics. *User's Manual. Model 336 Temperature Controller*. Rev. 1.8. Westerville, Ohio 43082-8888 USA, January 2014 (cit. on p. 69).

[33] Inc. Agilent Technologies. *User's Manual Agilent 3499A/B/C Switch/Control System*. Revision F. Loveland, Colorado, U.S.A., October 2012 (cit. on p. 94).

[34] Magnet-Physik Dr. Steingroever GmbH. *Betriebsanleitung FH 54 Gauss-/Teslameter*. BA - Nr.: 9920040201. Köln, Germany, June 2000 (cit. on p. 95).

[35] Ametek Advanced Measurement Technology, Inc. *Model 5210 Dual Phase Lock-in Amplifier. Instruction Manual*. 219874-A-MNL-G (cit. on p. 97).

[36] Philips Industrial & Electro-acoustic Systems. *PM 5193 Programmable syntesizer/function generator. Operating manual*. 9445 051 93001. Amsterdam, Netherlands (cit. on p. 104).

```
.->]<+++>-[---.-----------.++++++++++.>]<
+++>--[.>]<++>-[+++.->]<+>---[--.---.----
--------.>]<++++>-[---.-->]<--->++[..+>]<
+>---[.+>]<++>-[.+++>]<+++++>--[.+++.++++
+++++.>]<+++>-[++.+++>]<+>----[+.++++++++
.---.+++++++++++++.-------.->]<+++++>----
[.->]<+>---[->.->]<+>---[-.----------.+++
++++++++++.+>]<+++>-[.------------.>]<+++
>-[---.-------.-.>]<+>---[+.>]<++>-[+++++
+.----->]<+>---[.-->]<+>---[.-->]<+>---[-
```

RG91Z2xhcyBBZGFtcyA0MTk4NCk=