FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

FH Aachen

Faculty Electrical Engineering and Information Technology

Information Systems Engineering

Master Thesis

Development of a hardware and software framework for the automated characterization of permanent magnets for low-field MRI systemsg

Submitted by	**Marcel Werner Heinrich Friedrich Ochsendorf**
	Matriculation number: **3120232**
Examiner:	Prof. Dr.-Ing. Thomas Dey
Examiner:	Prof. Dr.-Ing. Volkmar Schulz
Date:	01.01.2024

Erklärung

I hereby declare that I have prepared this thesis independently and without outside assistance. Text passages, which are based literally or in the sense on publications or lectures of other authors, are marked as such. The work has not yet been submitted to any other examination authority and has not yet been published.

Aachen, _____, _____

Abstract

In the construction of low-field MRI devices based on permanent magnets, a large number of magnets are used. In order to realize a homogeneous B0 field with these magnets, which is necessary for many setups, the magnetic properties of these magnets have to be as similar to a certain degree. Due to the complex manufacturing process of neodymium magnets, the different properties, the direction of magnetization, can deviate from each other, which affects the homogeneity of the field. To adjust the field afterwards, a passive shimming process is typically performed, which is complex and time-consuming and requires manual corrections to the magnets used. To avoid this process, magnets can be systematically measured in advance. In this methodology, the recording, data storage and subsequent evaluation of the data play an important role. Various existing open-source solutions implement individual parts, but do not provide a complete data processing pipeline from aqusation to analysis and the data storage formats of these are not compatible to each other. For this use case, the MagneticReadoutProcessing library was created, which implements all major aspects of acquisition, storage, analysis, and each intermediate step can be customized by the user without having to create everything from scratch, favoring an exchange between different user groups. Complete documentation, tutorials and tests enable users to use and adapt the Framework as quickly as possible. The framework for the characterisation of different magnets, which requires the integration of magnetic field sensors, was used for the evaluation.

List of abbreviations

CLI Command Line Interface. 6, 15–17, 22

GUI Graphical User Interface. 15

HAL Hardware Abstraction Layer. 12

HTML Hypertext Markup Language. 23

I2C Inter-Integrated Circuit. 4

IDE Integrated development environment. 19, 22

IP Internet Protocol. 13

MRP MagneticReadoutProcessing. 10, 15–17, 19–23

PC Personal Computer. 10–13, 16, 20

PDF Portable Document Format. 23

PIP Python Package Installer. 20

PPS Puls Per Second. 13

PTP Precision Time Protocol. 13

REST Representational State Transfer. 10, 12

USB Universal Serial Bus. 13

Contents

1 Introduction

1.1 Background and Motivation

1.1.1 Low-Field MRI

1.1.2 Shimming procedure

1.2 Aim of this Thesis

1.3 Structure

2 State of the art

2.1 Opensource projects

2.2 Conceptual design

- etnwicklung eines hardware uns software framework zur einfachen aquirierung von Meagnetfelddaten
- analysetools und funktionen

3 Usecases

4 Unified Sensor

- ziel ist es einen low cost hallsensor-interface zu entwickeln welcher möglichst
- universell
- verschienee sensoren abbilden kann
- mit verschienden magneten typen und formen nutzbar
- reproduzierbar
- 1d, 2d, 3d
- integration

4.1 Sensor selection

Tabelle 4.1: Implemented digital halleffect sensors

	TLV493D-A1B6	HMC5883L	MMC5603NJ	AS5510
Readout-Axis	3D	3D	3D	1D
Temperature-Sensor	yes	no	yes	no
Resolution [uT]	98	0.2	0.007	48
Range [mT]	±130.0	±0.8	±3	±50
Interface	Inter-Integrated Circuit (I2C)	I2C	I2C	I2C

- for higher ranges analog sensoren nutzen welche jedoch eine aufwändigere schaltung erfodern
- datenblatte links ergänzen
- alle i2c in der regel, welches eine einfache integration ermöglicht
- eingebauter temperatursensor ermöglicht temperaturkompensation
- conrad teslameter mit seperaten temperatursensor

4.2 Mechanical Structure

- 3d druck toleranztest
- magnet halterung mit kraftloser arretierung
- motoren und andere unter umstaänden magnetische teile in der nähe des sensors
- nylon schrauben, 3d druck, 3d gedruckte klemmverbindungen
- später rausrechnen durch kalibierung

4.3 Electrical Interface

- usb, ethernet
- pps input output
- multiplexer for i2c sensors alredy implemented

4.4 Firmware

- automatic sensor detation ablaufdiagramm erst nach i2c geräte scannen dann analog versuchen

- serial cli support for manual mode

- sync impulse => 1 mastersensor als taktqelle

- interene mittelung und speichern der werte im buffer ringubber welcher bei jedem sync impuls neu belegt wird

```
help
=============================================================================================
> help                          shows this message
> version                       prints version information
> id                            sensor serial number for identification purposes
> sysstate                      returns current system state machine state
> opmode                        returns 1 if in single mode
> sensorcnt                    returns found sensorcount
> readsensor x <0-senorcount>  returns the readout result for a given sensor index for X axis
> readsensor y <0-senorcount>  returns the readout result for a given sensor index for Y axis
> readsensor z <0-senorcount>  returns the readout result for a given sensor index for Z axis
> readsensor b <0-senorcount>  returns the readout result for a given sensor index for B axis
> temp                          returns the system temperature
> anc <base_id>                 perform a autonumbering sequence manually
> ancid                         returns the current set autonumbering base id (-1 in singlemode)
> reset                         performs reset of the system
> info                          logs sensor capabilities
> commands                      lists sensor implemented commamnds which can be used by hal
=============================================================================================
```

Figure **4**-1: Sensors CLI

```
readsensor b 0
3279.99
```

Figure **4**-2: Query sensors b value using CLI

- 2. core übernimmt mittelung und auswetung

- was durch den user implementiert werden muss klasse

4.4.1 User-Interface

- einfache bedienung durch nutzer auch ohne weitere software
- configuration
- debugging

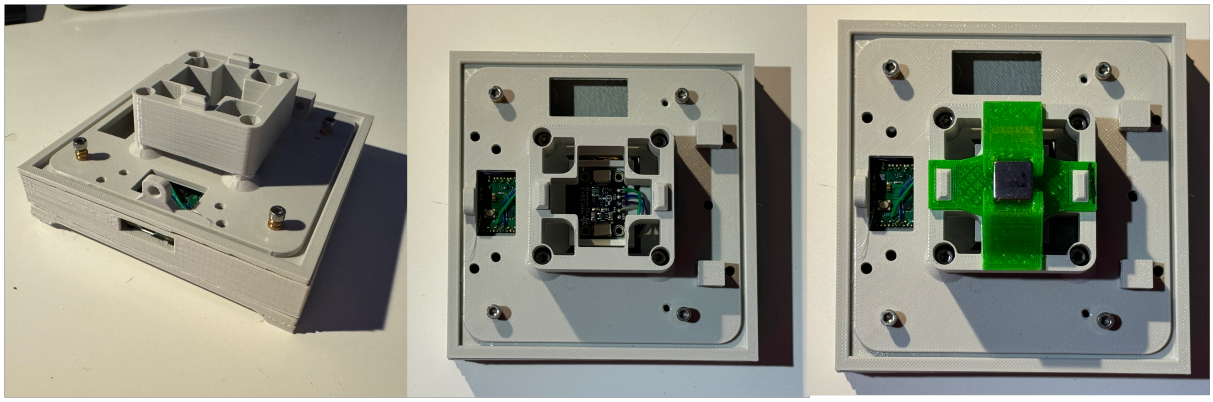

Figure **4**-3: 1D sensor contrsuction with universal magnet mount

4.5 Sensor Syncronsisation

4.6 Example Sensors

anbei werden drei erschienee sensoren für unterschiedliche anwedungfälle tablle statisch dynamisch

4.6.1 1D: Single Sensor

- einfachster aufbau rp pico + sensor

4.6.2 1D: Dual Sensor

- gleicher abstand zwei gleicher sensoren
- schnelle erkennung der plarisationsebene ggf offset vom mittelpunkt dieser

4.6.3 Full-Sphere

- komplexester aufbau sensor + mechanik
- polar mechanisches system
- full sphere sensor

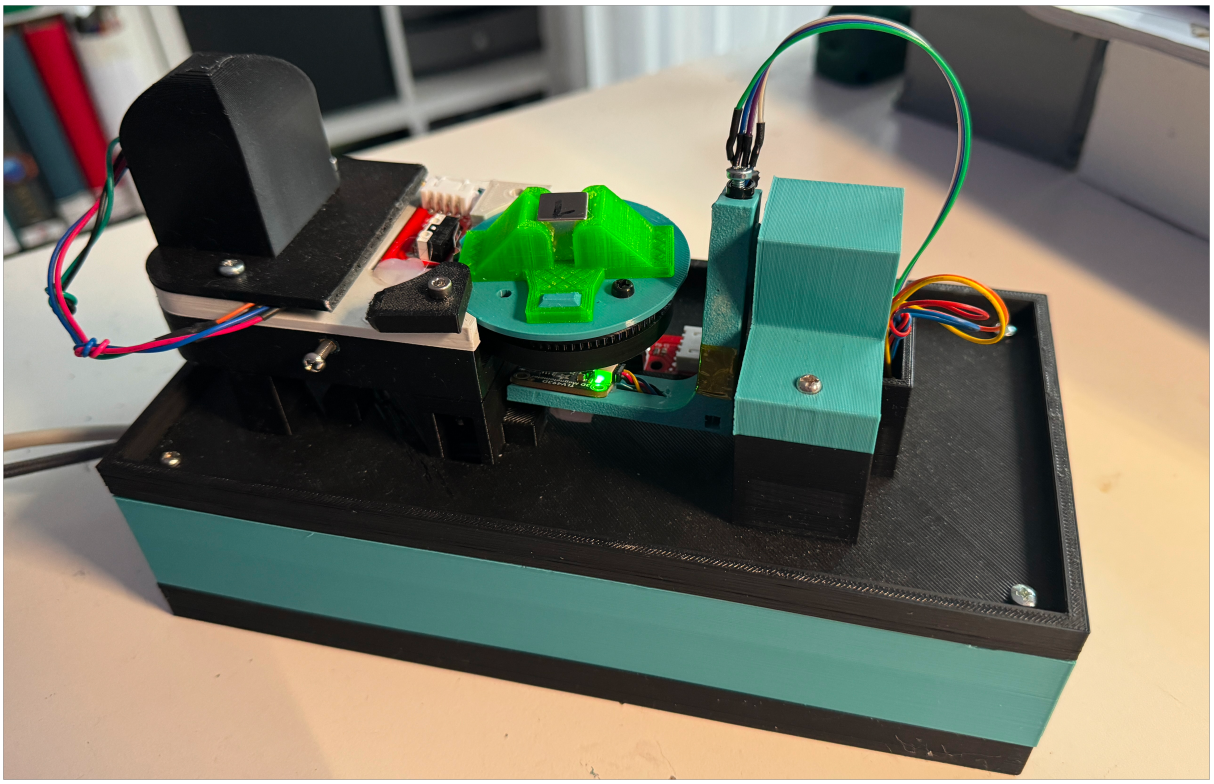

Figure **4**-4: Full-Sphere sensor implementation using two Nema17 stepper motors in a polar coordinate system

4.6.4 Integration of an Professional Teslameter

- einfach anbindung professioneller teslameter
- Voltkraft

5 Software readout framework

5.1 Library requirements

5.1.1 Concepts

- beispiele für projekte welche nur einzelne schnritte implementieren
- so kann man sich auf die implementierung

5.1.2 User interaction points

- grafik zeigen
- einzelne module erleutern

HAL

- aufbau hal im grunde wird nur ein die commandos an das sensor cli weitergegeben
- alle sensoren implementieren mehr oder weniger die gleichen befehle
- hal gibt nur weiter und ist “dumm”

Visualisation

5.1.3 Export

- format import export

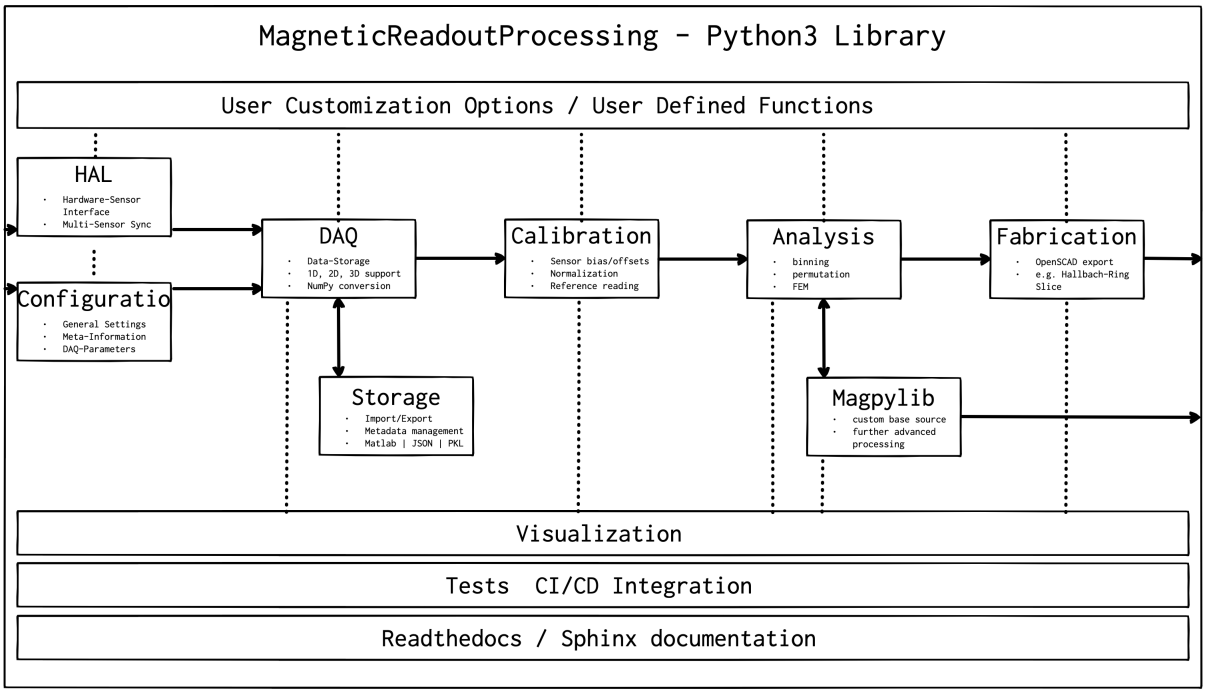

Figure **5**-1: MRPlib COMPLETE FLOW

- matlab

Meta-Data

5.1.4 Multi-Sensor setup

At the current state of implementation, it is only possible to detect and use sensors that are directly connected to the Personal Computer (PC) with the MagneticReadoutProcessing (MRP)-library. This has the disadvantage that there must always be a physical connection. This can make it difficult to install multiple sensors in measurement setups where space or cable routing options are limited. To make sensors connected to a small remote PC available on the network, the Proxy**5**-2 module has been developed. This can be a single board computer (e.g. a Raspberry Pi). The small footprint and low power consumption make it a good choice. It can also be used in a temperature chamber. The approach of implementing this via a Representational State Transfer (REST) interface also offers the advantage that several measurements or experiments can be recorded at the same time with the sensors.

Another application example is when sensors are physically separated or there are long distances between them. By connecting several sensors via the proxy module, it is

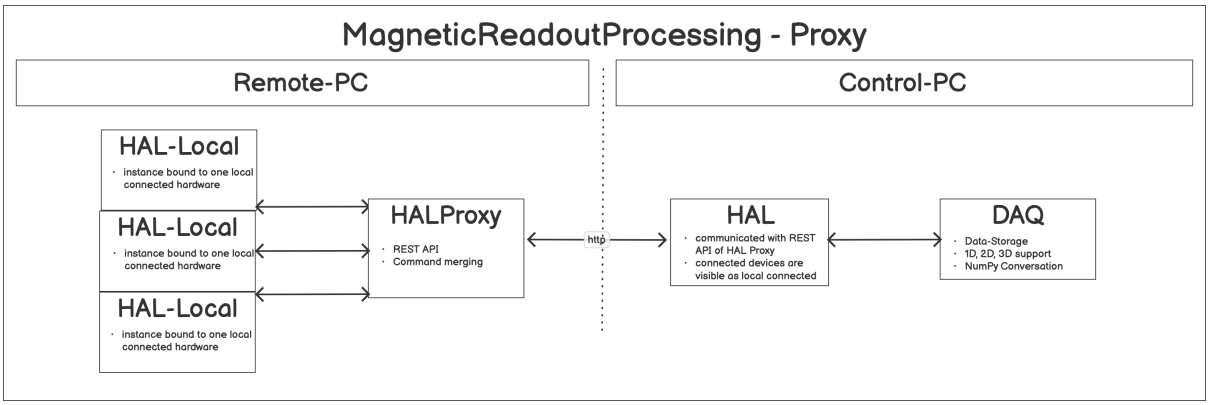

Figure **5**-2: MRPlib Proxy Module

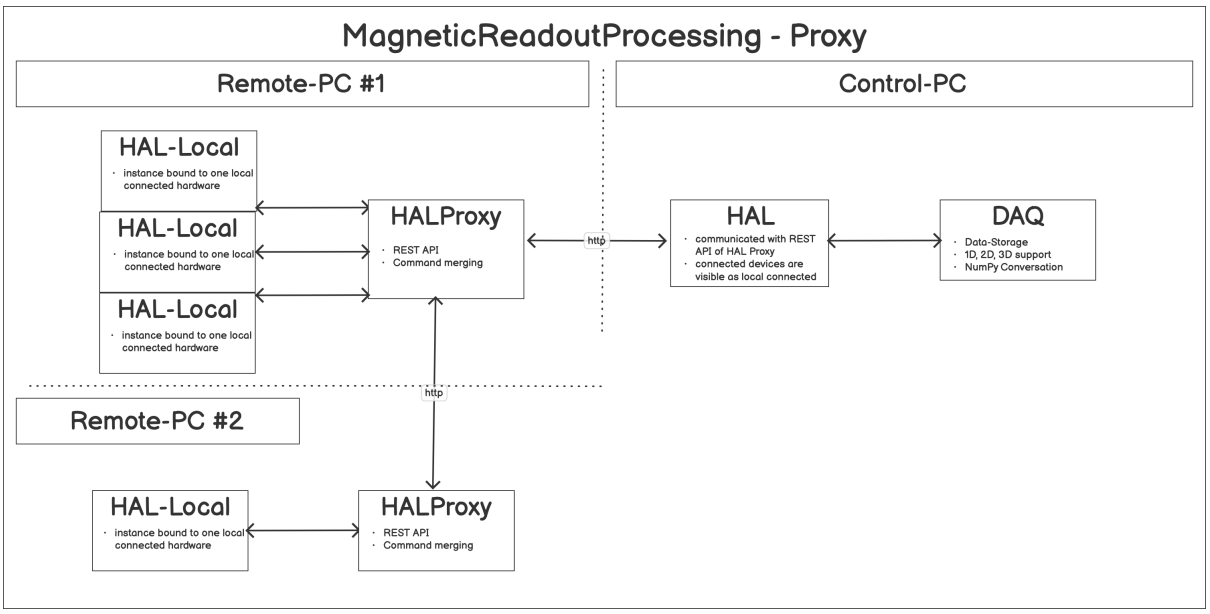

Figure **5**-3: mrp proxy multi

possible to link several instances and all sensors available in the network are available to the control PC.

The figure **5**-3 shows the modified multi–proxy – multi–sensor topology. Here, both proxy instances do not communicate directly with the control PC, but remote (+pc) #2 is connected to remote (+pc) #1. This is then visible as a sensor opposite the Control PC, even if there are several proxy instances behind it.

Network-Proxy

The figure **5**-2 shows the separation of the various Hardware Abstraction Layer (HAL) instances, which communicate with the physically connected sensors on the remote-PC and the control-PC side, which communicates with the remote side via the network. For the user, nothing changes in the procedure for setting up a measurement. The proxy application must always be started5.1 on the remote PC side.

```
# START PROXY INSTNACE WITH TWO LOCALLY CONNECTED SENSORS
$ python3 mrpproxy.py proxy launch /dev/ttySENSOR_A /dev/ttySENSOR_B # add another proxy instance http://proxyinstance_2.local for multi-sensor, multi-proxy chain
Proxy started. http://remotepc.local:5556/
PRECHECK: SENSOR_HAL: 1337 # SENSOR A FOUND
PRECHECK: SENSOR_HAL: 4242 # SENSOR B FOUND
Terminate  Proxy instance [y/N] [n]:
```

Listing 5.1: MRPproxy usage to enable local sensor usage over network

After the proxy instance has been successfully started, it is optionally possible to check the status via the REST interface:5.2

```
# GET PROXY STATUS
$ wget http://proxyinstance.local:5556/proxy/status
{
"capabilities":[
  "static",
  "axis_b",
  "axis_x",
  "axis_y",
  "axis_z",
  "axis_temp",
  "axis_stimestamp"
],
"commands":[
  "status",
  "initialize",
  "disconnect",
  "combinedsensorcnt",
  "sensorcnt",
  "readsensor",
  "temp"
]

# RUN A SENSOR COMMAND AND GET THE TOTAL SENSOR COUNT
```

```
24    $ wget http://proxyinstance.local:5556/proxy/command?cmd=
      combinedsensorcnt
25    {
26    "output":[
27       "2"
28    ]
29 }
30 }
```

Listing 5.2: MRPproxy REST enpoiint query examples

The query result shows that the sensors are connected correctly and that their capabilites have also been recognised correctly. To be able to configure a measurement on the other, only the Internet Protocol (IP) address or host name of the remote PC is required5.3.

```
1 # CONFIGURE MEASUREMENT JOB USING A PROXY INSTANCE
2 $ MRPcli config setupsensor testcfg --path http://proxyinstance.local
     :5556
3 > remote sensor connected: True using proxy connection:
4 > http://proxyinstance.local:5556 with 1 local sensor connected
```

Listing 5.3: MRPcli usage example to connect with a network sensor

Sensor Syncronisation

Another important aspect when using several sensors via the proxy system is the synchronisation of the measurement intervals between the sensors. Individual sensor setups do not require any additional synchronisation information, as this is communicated via the Universal Serial Bus (USB) interface. If several sensors are connected locally, they can be connected to each other via their sync input using short cables. One sensor acts as the central clock (see chapter 4.5). this no longer works for long distances and the syncronisation must be made via the network connection.

If time-critical synchronisation is required, Precision Time Protocol (PTP) and Puls Per Second (PPS) output functionality can be used on many single-board computers, such as the RaspberryPi Compute Module.

- was ptp, bild pps output
- alle clients über ptp verbunden
- dso bild von jeff gerling über rpi4 ptp

Command-Router

- nummerieierung zuerst lokale sensoren dann weitere proxy sensoren
- commando templating

5.1.5 Examples

6 Usability improvements

Usability improvements in software libraries are crucial for efficient and user-friendly development. Intuitive API documentation, clearly structured code examples and improved error messages promote a smooth developer experience. Standardised naming conventions and well thought-out default values simplify the application. A Graphical User Interface (GUI) or CLI application for complex libraries can make it easier to use, especially for developers with less experience. Continuous feedback through automated tests and comprehensive error logs enable faster bug fixing. The integration of community feedback and regular updates promotes the adaptability of the MRP-library to changing needs. Effective usability improvements help to speed up development processes and increase the satisfaction of the developer community. In the following, some of these have been added in and around the MRP-library, but they are only optional components for the intended use.

6.1 Command Line Interface

In the first version of this MRP-library, the user had to write his own Python scripts even for short measurement and visualisation tasks. However, this was already time-consuming for reading out a sensor and configuring the measurement parameters and metadata and quickly required more than 100 lines of new Python code. Although

Figure **6**-1: MRP CLI output to configure a new measurement

such examples are provided in the documentation, it must be possible for programming beginners in particular to use them. To simplify these tasks, a CLI**6**-2 was implemented around this MRP-library, which is then also supplied as a fixed component. This CLI implements the following functionalities:

- Detection of connected sensors
- Configuration of measurement series
- Recording of measured values from stored measurement series
- Simple commands for checking recorded measurement series and their data.

Thanks to this functionality of the CLI, it is now possible to connect a sensor to the PC, configure a measurement series with it and run it at the end. The result is then an exported file with the measured values. These can then be read in again with the MRP-library and processed further. The bash code6.1 shows this procedure in detail.

```
1 # CLI EXAMPLE FOR CONFIGURING A MEASUREMENT RUN
2 ## CONFIGURE THE SENSOR TO USE
3 $ MRPcli config setupsensor testcfg
4 > 0 - Unified Sensor 386731533439 - /dev/cu.usbmodem3867315334391
5 > Please select one of the found sensors [0]:
6 > sensor connected: True 1243455
7 ## CONFIGURE THE MEASUREMENT
8 $ MRPcli config setup testcfg
9 > CONFIGURE testcfg
10 > READING-NAME: [testreading]: testreading
11 > OUTPUT-FOLDER [/cli/reading]: /tmp/reading_folder_path
12 > NUMBER DATAPOINTS: [1]: 10
13 > NUMBER AVERAGE READINGS PER DATAPOINT: [1]: 100
14 # RUN THE CONFIGURED MEASUREMENT
15 $ MRPcli measure run
16 > STARTING MEASUREMENT RUN WITH FOLLOWING CONFIGS: ['testcfg']
17 > config-test: OK
18 > sensor-connection-test: OK
19 > START MEASUREMENT CYCLE
20 > sampling 10 datapoints with 100 average readings
21 > SID:0 DP:0 B:47.359mT TEMP:23.56
22 > ....
23 > dump_to_file testreading_ID:525771256544952_SID:0_MAG:
     N45_CUBIC_12x12x12.mag.json
```

Listing 6.1: CLI example for configuring a measurement run

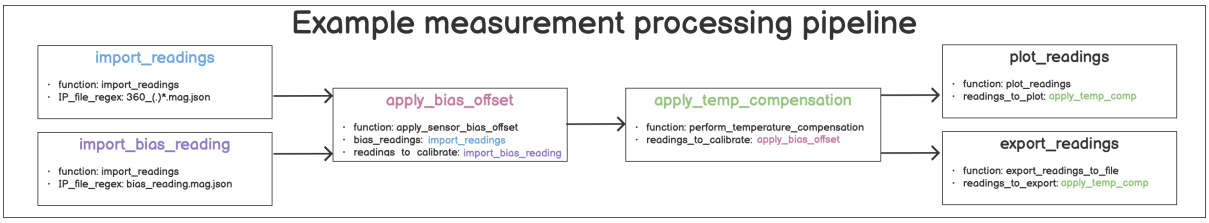

Figure **6**-2: example measurement analysis pipeline

6.2 Programmable data processing pipeline

After it is very easy for users to carry out measurements using the CLI, the next logical step is to analyse the recorded data. This can involve one or several hundred data records. Again, the procedure for the user is to write their own evaluation scripts using the MRP-library. This is particularly useful for complex analyses or custom algorithms, but not necessarily for simple standard tasks such as bias compensation or graphical plot outputs. For this purpose, a further CLI application was created, which enables the user to create and execute complex evaluation pipelines for measurement data without programming. The example**6**-2 shows a typical measurement data analysis pipeline, which consists of the following steps:

- Import the measurements
- Determine sensor bias value from imported measurements using a reference measurement
- Apply linear temperature compensation
- Export the modified measurements
- Create a graphical plot of all measurements with standard deviation

In order to implement such a pipeline, the yaml file format was chosen for the definition of the pipeline, as this is easy to understand and can also be easily edited with a text editor. Detailed examples can be found in the documentation[2]. The pipeline definition consists of sections which execute the appropriate Python commands in the background. The signatures in the yaml file are called using reflection and a real-time search of the loaded global() symbol table[6]. This system makes almost all Python functions available to the user. To simplify use, a pre-defined list of tested MRP library functions for use in pipelines is listed in the documentation[2]. The following pipeline definition6.2 shows the previously defined steps**6**-2 as yaml syntax.

```
1 stage import_readings:
2   function: import_readings
3   parameters:
```

```
4     IP_input_folder: ./readings/fullsphere/
5     IP_file_regex: 360_(.)*.mag.json
6
7 stage import_bias_reading:
8   function: import_readings
9   parameters:
10    IP_input_folder: ./readings/fullsphere/
11    IP_file_regex: bias_reading.mag.json
12
13 stage apply_bias_offset:
14   function: apply_sensor_bias_offset
15   parameters:
16    bias_readings: stage import_bias_reading
17    readings_to_calibrate: stage import_readings
18
19 stage apply_temp_compensation:
20   function: apply_temperature_compensation
21   parameters:
22    readings_to_calibrate: stage import_readings
23
24 stage plot_normal_bias_offset:
25   function: plot_readings
26   parameters:
27    readings_to_plot: stage apply_temp_compensation
28    IP_export_folder: ./readings/fullsphere/plots/
29    IP_plot_headline_prefix:  Sample N45 12x12x12 magnets calibrated
30
31 stage export_readings:
32   function: export_readings
33   parameters:
34    readings_to_plot: stage apply_temp_compensation
35    IP_export_folder: ./readings/fullsphere/plots/
```

Listing 6.2: Example User Defined Processing Pipeline

- blöcke erleutern insbesondere stage
- wie calltree gebildet

6.3 Tests

Software tests in libraries offer numerous advantages for improving quality and efficiency. Firstly, they enable the identification of errors and vulnerabilities before software is

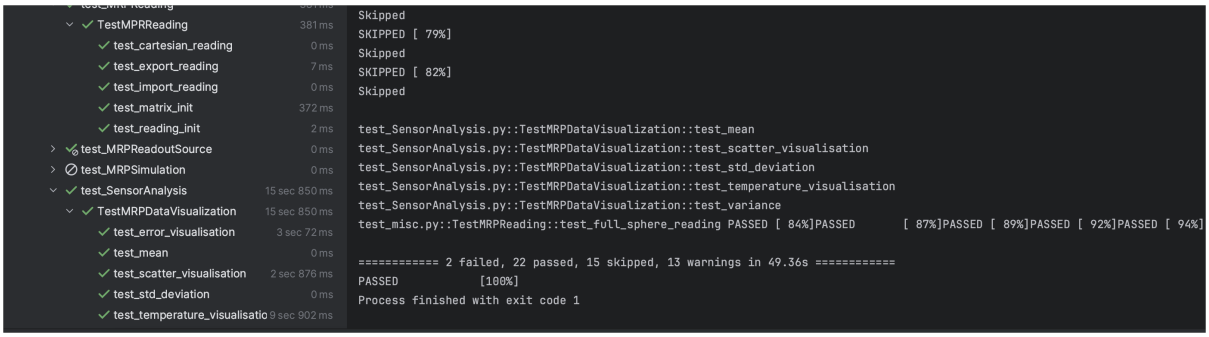

Figure **6**-3: MRP library test results for different submodules executed in PyCharm IDE

published as a new version. This significantly improves the reliability of MRP-library applications. Tests also ensures consistent and reliable performance, which is particularly important when libraries are used by different users and for different usecases.

During the development of the MRP-library, test cases were also created for all important functionalities and use cases. The test framework PyTest[9] was used for this purpose, as it offers direct integration in most IDEs (see **6**-3) and also because it provides detailed and easy-to-understand test reports as output in order to quickly identify and correct errors. It also allows to tag tests, which is useful for grouping tests or excluding certain tests in certain build environment scenarios. Since all intended use cases were mapped using the test cases created, the code of the test cases could later be used in slightly simplified variants6.3 as examples for the documentation.

```python
# $ cat test_mrpreading.py
class TestMPRReading(unittest.TestCase):
    # PREPARE A INITIAL CONFIGURATION FILE FOR ALL FOLLOWING TEST CASES IN THIS FILE
    def setUp(self) -> None:
        self.test_folder: str = os.path.join(os.path.dirname(os.path.abspath(__file__)), "tmp")
        self.test_file:str = os.path.join(self.import_export_test_folderpath, "tmp")

    def test_matrix(self):
        reading: MRPReading = MRPSimulation.generate_reading()
        matrix: np.ndarray = reading.to_numpy_matrix()
        n_phi: float = reading.measurement_config.n_phi
        n_theta: float = reading.measurement_config.n_theta
        # CHECK MATRIX SHAPE
        self.assertTrue(matrix.shape != (n_theta,))
        self.assertTrue(len(matrix.shape) <= n_phi))

    def test_export_reading(self) -> None:
        reading: MRPReading = self.test_reading_init()
```

```
19          self.assertIsNotNone(reading)
20          # EXPORT READING TO A FILE
21          reading.dump_to_file(self.test_file)
22
23      def test_import_reading(self):
24          # CREATE EMPTY READING
25          reading_imported:MRPReading = MRPReading.MRPReading(None)
26          # LOAD READING FROM FILE
27          reading_imported.load_from_file(self.test_file)
28          # CHECK IF ENTRIES ARE POPULATED
29          self.assertIsNotNone(reading_imported.additional_data)
30          self.assertIsNotNone(reading_imported.data)
```

Listing 6.3: Example pytest class for testing MRPReading module functions

One problem, however, is the parts of the MRP-library that require direct access to external hardware. These are, for example, the MRPHal and MRPHalRest modules, which are required to read out sensors connected via the network. Two different approaches were used here. In the case of local development, the test runs were carried out on a PC that can reach the network hardware and thus the test run could be carried out with real data.

In the other scenario, the tests are to be carried out before a new release in the repository on the basis of Github Actions[8]. Here there is the possibility to host local runner software, which then has access to the hardware, but then a PC must be permanently available for this task. Instead, the hardware sensors were simulated by software and executed via virtualisation on the systems provided by Github Actions[8].

6.4 Package distribution

One important point that improves usability for users is the simple installation of the MRP-library. As it was created in the Python programming language, there are several public package directories where users can provide their software modules. Here, PyPi [5]**6**-4[4] is the most commonly used package directory and offers direct support for the package installation programm Python Package Installer (PIP)6.4.

In doing so, PIP not only manages possible package dependencies, but also manages the installation of different versions of a package. In addition, the version compatibility is also checked during the installation of a new package, which can be resolved manually

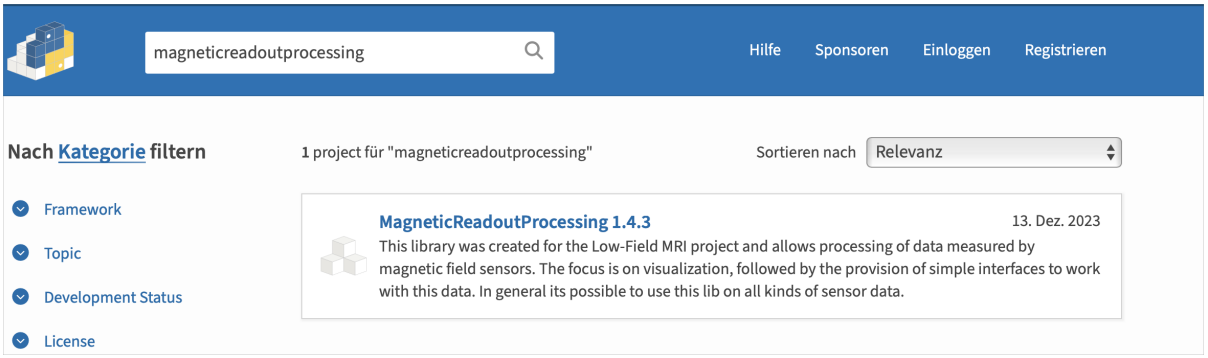

Figure **6**-4: MagneticReadoutProcessing library hosted on PyPi

by the user in the event of conflicts.

```
# https://pypi.org/project/MagneticReadoutProcessing/
# install the latest version
$ pip3 install MagneticReadoutProcessing
# install the specific version 1.4.0
$ pip3 install MagneticReadoutProcessing==1.4.0
```

Listing 6.4: Bash commands to install the MagneticReadoutProcessing (+mrp)-library using pip

To make the MRP-library compatible with the package directory, Python provides separate installation routines that build a package in an isolated environment and then provide an installation wheel archive. This can then be uploaded to the package directory.

Since the MRP-library requires additional dependencies (e.g. numpy, matplotlib), which cannot be assumed to be already installed on the target system, these must be installed prior to the actual installation. These can be specified in the MRP-library installation configuration setup.py6.5 for this purpose.

```
# dynamic requirement loading using 'requirements.txt'
req_path = './requirements.txt'
with pathlib.Path(req_path).open() as requirements_txt:
    install_requires = [str(requirement) for requirement in
    pkg_resources.parse_requirements(requirements_txt)]

setup(name='MagneticReadoutProcessing',
      version='1.4.3',
      url='https://github.com/LFB-MRI/MagnetCharacterization/',
      packages=['MRP', 'MRPcli', 'MRPudpp', 'MRPproxy'],
      install_requires=install_requires,
      entry_points={
          'console_scripts': [
```

```
13              'MRPCli = MRPcli.cli:run',
14              'MRPUdpp = MRPudpp.uddp:run',
15              'MRPproxy = MRPproxy.mrpproxy:run'
16          ]
17      }
18  )
```

Listing 6.5: setup.py with dynamic requirement parsing used given requirements.txt

To make the CLI scripts written in Python easier for the user to execute without having to use the python3 prefix. This has been configured in the installation configuration using the entry_points option, and the following commands are available to the user:

- MRPcli --help instead of python3 cli.py --help
- MRPudpp --help instead of python3 udpp.py --help
- MRPproxy --help instead of python3 proxy.py --help

In addition, these commands are available globally in the system without the terminal shell being located in the MRP-library folder.

6.4.1 Documentation

In order to provide comprehensive documentation for the enduser, the source code was documented using Python-docstrings[7] and the Python3.5 type annotations:

- Function description
- Input parameters - using param and type
- Return value - using returns, rtype

The use of type annotations also simplifies further development, as modern IDEs can more reliably display possible methods to the user as an assistance.**??**

```
1   # MRPDataVisualisation.py - example docstring
2   def plot_temperature(_readings: [MRPReading.MRPReading], _title:
    str = '', _filename: str = None, _unit: str = "degree C") -> str:
3       """
4       Plots a temperature plot of the reading data as figure
5       :param _readings: readings to plot
6       :type _readings: list(MRPReading.MRPReading)
7       :param _title: Title text of the figure, embedded into the head
```

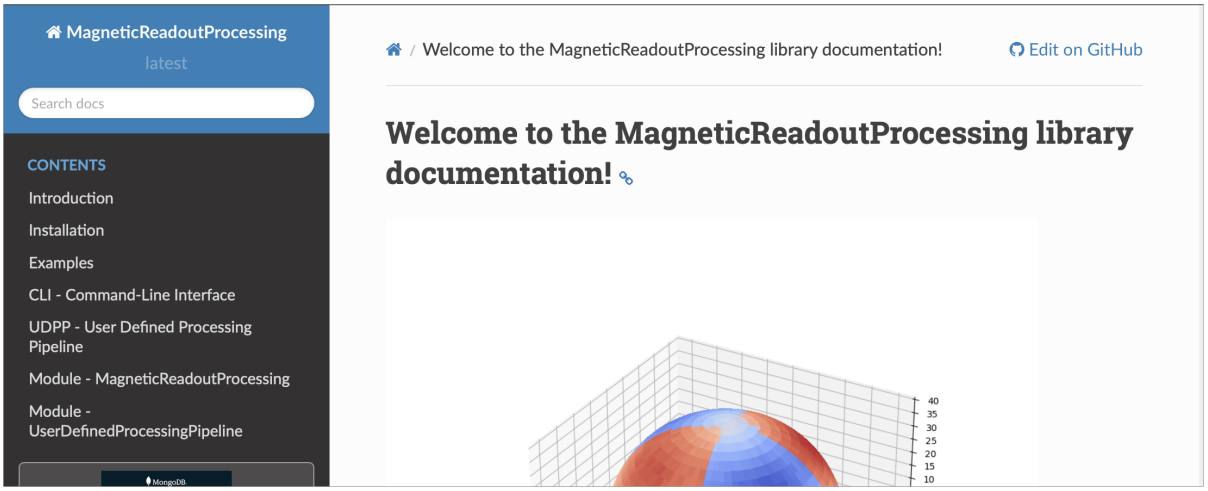

Figure **6**-5: MagneticReadoutProcessing documentation hosted on ReadTheDocs

```
8         :type _title: str
9         :param _filename: export graphic to an given absolute filepath
      with .png
10        :type _filename: str
11        :returns: returns the abs filepath of the generated file
12        :rtype: str
13        """
14        if _readings is None or len(_readings) <= 0:
15            raise MRPDataVisualizationException("no readings in
      _reading given")
16        num_readings = len(_readings)
17        # ...
```

Listing 6.6: Python docstring example

Since ‘docstrings’ only document the source code, but do not provide simple how-to-use instructions, the documentation framework Sphinx[1] was used for this purpose. This framework makes it possible to generate Hypertext Markup Language (HTML) or Portable Document Format (PDF) documentation from various source code documentation formats, such as the used docstrings. These are converted into a Markdown format in an intermediate step and this also allows to add further user documentation such as examples or installation instructions.

In order to make the documentation created by Sphinx accessible to the user, there are, as with the package management by PyPi services, which provide Python MRP-library documentation online.

Once the finished documentation has been generated from static HTML files, it is stored

in the project repository. Another publication option is to host the documentation via online services such as ReadTheDocs[3], where users can make documentation for typical software projects available to others.

The documentation has also been uploaded for ReadTheDocs[2] and linked in the repository and on the overview page**6**-5 on PyPi.

The process of creating and publishing the documentation has been automated using GitHub Actions[8], so that it is always automatically kept up to date with new features.

7 Evaluation

7.1 Prequesites for evaluation

7.2 Evaluation confiugration

7.2.1 Sensor readout

7.2.2 Processing pipeline

7.3 Test scenarios

7.4 Results

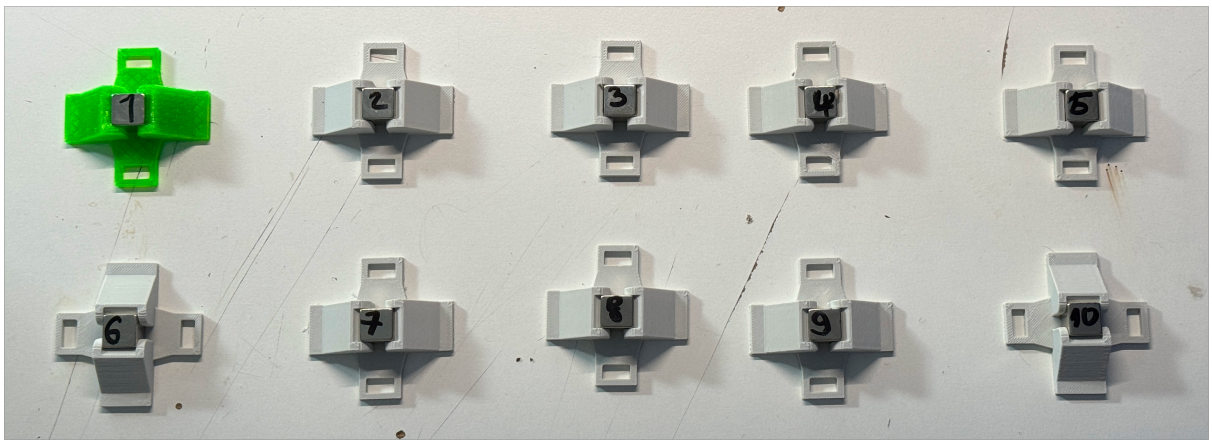

Figure **7**-1: testmagnets in holder

8 Conclusion and dicussion

8.1 Conclusion

8.2 Problems

8.3 Outlook

- magfield camera

Literaturverzeichnis

[1] DEVELOPERS, Sphinx: *Sphinx makes it easy to create intelligent and beautiful documentation.* https://www.sphinx-doc.org/en/master/. Version: 01.01.2024

[2] DOCS, Inc Read t.: *MagneticReadoutProcessing - ReadTheDocs*. https://magneticreadoutprocessing.readthedocs.io/en/latest/index.html. Version: 01.01.2024

[3] DOCS, Inc Read t.: *ReadTheDocs - Build, host, and share documentation, all with a single platform.* https://readthedocs.com. Version: 01.01.2024

[4] FOUNDATION, Python S.: *MagneticReadoutProcessing - PyPi*. https://pypi.org/project/MagneticReadoutProcessing/. Version: 01.01.2024

[5] FOUNDATION, Python S.: *PyPi - The Python Package Index (PyPI) is a software directory for the Python programming language.* https://pypi.org. Version: 01.01.2024

[6] FOUNDATION, Python S.: *Python Docs - globals*. https://docs.python.org/3/library/functions.html#globals. Version: 01.01.2024

[7] FOUNDATION, Python S.: *Python Enhancement Proposals - Docstring Conventions*. https://peps.python.org/pep-0257/. Version: 01.01.2024

[8] GITHUB, Inc.: *Github Actions - Automate your workflow from idea to production*. https://github.com/features/actions. Version: 01.01.2024

[9] KREKEL, Holger ; TEAM pytest-dev: *pytest: helps you write better programs*. https://docs.pytest.org/en/7.4.x/. Version: 01.01.2024

List of Figures

List of Tables

Listings