ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Supervisory Control and Data Acquisition system of the n2EDM experiment

Master Thesis

Konstantin Nesterov

Monday 16th December, 2019

Advisors: Prof. Dr. K. S. Kirch, Dr. J. Krempel

Department of Physics, ETH Zürich

Abstract

Just as nowadays no serious experiment can be built and conducted by a single person, the experiment itself cannot consist of a single tool. The n2EDM experiment aims to achieve an ambitious goal: to measure the electric dipole moment of a neutron with a new level of precision. Such challenging project demands the need for the complex and well-connected system. This thesis intends to describe the development of new and improvement of existing components, such as:

- **COM handler** — an adapter translating the POSIX pipes to the TCP/IP connections. Almost every node in the DAQ system is connected with others through it.
- **Sequencer** — a software node orchestrating other nodes. It follows the user-generated script allowing one to describe the reproducible behaviour of the whole DAQ system with a human-readable set of commands.
- **Proxy for the remote magnetometers** — a smart bridge between the pool of the remote magnetometers and a standard TCP/IP interface of the COM handler.
- **Surrounding field compensation system** — a system for the active stabilisation of the magnetic field. It uses the a set of controlled coils to minimise the magnetic fluctuations in the area of the n2EDM experiment.

These pieces are essential for the n2EDM experiment to function, so the aim was to make them error-resistant, extendable and easy to support for the future developers.

Check the source at https://github.com/wasd171/diploma-msc

Acknowledgements

I would like to express my gratitude to everyone whose contributions and involvement made this thesis possible:

- **Klaus Kirch** for providing an opportunity of working in the Precision Physics at Low Energy group and always being kind and thoughtful.
- **Jochen Krempel** and **Dieter Ries** for being open to new ideas, trusting my vision and always being very helpful with all the questions about the n2EDM DAQ system that I had.
- **James Margrove** and **Simon Hofer** for backing me up every day.
- **Krovostok** for their comforting beats that boosted my productivity.
- My family with special thanks to my mother **Elena Nesterova** for her unconditional love and support over the whole course of my studies.
- And finally I want to give the most sincere and touching recognition to **Oxana Spirina** for always believing in me and being caring and inspiring during the toughest moments.

Contents

List of acronyms

- nEDM — *neutron Electric Dipole Moment*.
- PSI — *Paul Scherrer Institute*.
- UCN — *UltraCold Neutron*.
- ILL — *Institut Laue–Langevin*.
- HV — *High Voltage*.
- SFC — *Surrounding Field Compensation*.
- DAQ — *Data AcQuisition and control*.
- VI — *Virtual Instrument*.
- TCP/IP — *Transmission Control Protocol/Internet Protocol*.
- SCPI — *Standard Commands for Programmable Instruments*.
- E2E — *End-to-End*.
- POSIX — *Portable Operating System Interface*.
- IPC — *InterProcess Communication*.
- COM Handler — *COMmunication Handler*, see Section 3.6.
- PTP — *Precision Time Protocol*.
- GPS — *Global Positioning System*.
- GUI — *Graphical User Interface*.
- FIFO — *First In, First Out*, often short for the *FIFO POSIX pipes*.
- FFT — *Fast Fourier Transform*.
- RFC — *Request For Comments*.
- RM-proxy — *Remote Magnetometers' Proxy*, see Section 4.8.

Chapter 1

Introduction

Out of S. Okubo's effect
At high temperature
A fur coat is sewed for the Universe
Shaped for its crooked figure

A. D. Sakharov [25]

We interact with matter every day. Even you, the reader, are probably made out of matter! However, antimatter is so rare that it is considered to cost a few hundred millions Swiss francs per gram [7], making it the most expensive substance in the universe. Why does such a stunning difference in the abundance exist?

First step to solving this problem is to define what are the required conditions that would allow the disbalance to evolve. Those conditions [9] were described [25] by Andrei Sakharov in 1967:

- Violation of baryon number conservation.
- *C*- and *CP*-symmetry violation.
- Processes take place far from thermal equilibrium.

Let us take a look at the *CP*-symmetry and prove that a non-zero electric dipole moment of an elementary particle would indeed break it. We would select neutron as a particle of choice.

The neutron in the ground state has spin of $I = 1/2$ and can be characterised completely by a single quantum number of a spin projection $m_I = \pm 1/2$. We can write down a Hamiltonian [13] of this neutron in external electric and magnetic fields $\vec{E}$ and $\vec{B}$:

$$\mathcal{H} = -\frac{d_n \vec{I} \cdot \vec{E} + \mu_n \vec{I} \cdot \vec{B}}{I} \tag{1.1}$$

with d_n and μ_n being the electric and magnetic moments of the neutron [14].

It does not make sense to discuss the potential violation of the symmetries before we define them. Fundamental symmetries are blended into the fabric of our Universe by providing sufficient conditions [18] for the conservation laws. In our analysis we would consider three symmetries of the Standard Model: C, P and T.

- (C)harge —- replaces every particle with its antiparticle: $q \to -q$.
- (P)arity — inverts the physical space: $\vec{r} \to -\vec{r}$.
- (T)ime — turns the time back: $t \to -t$.

How would the P and T inversions affect [9] the Hamiltonian from Eq. 1.1?

Parity transformation only act on a polar vector of the electric field: $\vec{E} \to -\vec{E}$, both $\vec{B}$ and $\vec{I}$ are conserved. This brings us to

$$P\mathcal{H} = -\frac{d_n \vec{I} \cdot \left(-\vec{E}\right) + \mu_n \vec{I} \cdot \vec{B}}{I} \neq \mathcal{H}. \tag{1.2}$$

Time reversal would only affect the axial vectors $\vec{B}$ and $\vec{I}$: $\vec{B} \to -\vec{B}$, $\vec{I} \to -\vec{I}$, the field $\vec{E}$ is left as is:

$$T\mathcal{H} = -\frac{d_n \left(-\vec{I}\right) \cdot \vec{E} + \mu_n \left(-\vec{I}\right) \cdot \left(-\vec{B}\right)}{I} \neq \mathcal{H}. \tag{1.3}$$

Assuming that the CPT invariance [26] is conserved we derive the violation of a CP-symmetry which provides us the motivation to measure the EDM of a neutron.

"Wait a minute," could have said an attentive reader at this point. "Does not Standard Model predict a non-zero EDM of the neutron already? I am still not convinced why would you want to conduct this experiment."

And an attentive reader would have had a completely fair point! Indeed, Standard Model predicts [16] the following:

$$d_n \approx 2 \cdot 10^{-32}\ e \cdot \text{cm}. \tag{1.4}$$

However, we would still like to measure d_n for the reasons listed below:

- The only way to prove the theory is to check it experimentally. So far no one has measured d_n with a precision close to the predicted value.

- The result that can be achieved by using Standard Model is too weak to explain the baryogenesis [9], yet baryogenesis has clearly happened.
- If we go beyond Standard Model to find a mechanism through which the Universe as we know it could have been formed, we need to cut off theories that do not agree with experimental data. This is something that this experiment does perfectly: on the Fig. 11 one can see all theoretical models that the measurement of the neutron EDM has ruled out, allowing the scientists to focus on more prominent leads.

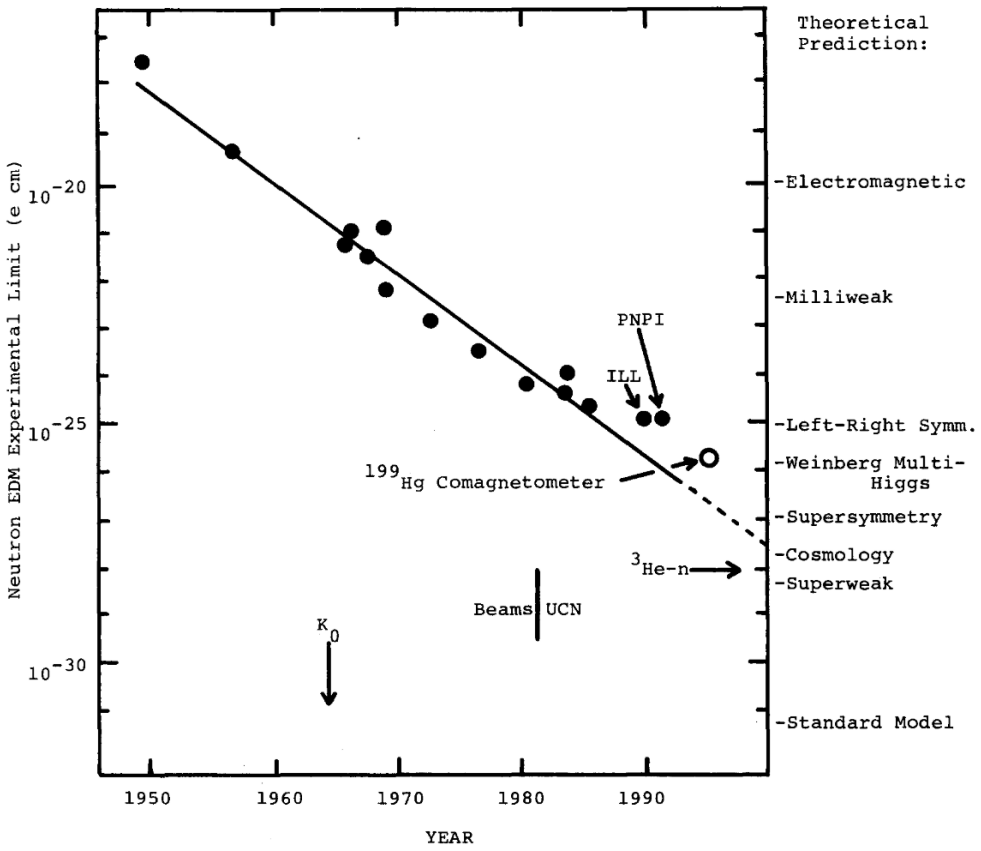

Figure 11: Measurement history of the neutron EDM, graph taken from [13].

Hopefully these reasons would convince even the most demanding reader in the need to conduct the n2EDM experiment. But what is *n2EDM* exactly? We will try to explain that in the next chapter.

Chapter 2

The n2EDM experiment

Knowledge is power and our knowledge of the physical property that can be measured to show the violation of the *CP*-symmetry is an important first step in solving the riddle of the baryogenesis. Now we just need to get our hands dirty with some experimental data. It will be obtained over the course of the n2EDM experiment currently being built at PSI (Paul Scherrer Institute, Villigen, Switzerland).

What will be measured? Electric dipole moment of the neutron. The neutrons were chosen for the following reasons:

- They are electrically neutral[1], which means that they would not be dragged by the electric field $\vec{E}$.
- There are nuclear reactions that allow to produce them efficiently, like fission or spallation (which is already available in PSI and will be used).
- They can be cooled down to become UCNs (ultracold neutrons).

What are ultracold neutrons and why do we like them? We call [10] a neutron ultracold when it has a kinetic energy $E_{kin} \leq 300$ neV. Such low energy brings the following experimental benefits:

- Ease of collection, since the neutrons would behave similar to ping-pong balls, bouncing from the surface of a neutron vessel.
- Possibility to store [31] the neutrons up to their lifetime of ≈ 886 s [28].
- Weakening of a so-called $\vec{v} \times \vec{E}$ effect [20], which arises from the coupling of a particle spin $\vec{I}$ itself with an electric field $\vec{E}$. This would bring the effective Hamiltonian closer to one mentioned in Eq. 1.1.

[1] Which is quite important for an experiment based in Switzerland

What precision do we expect? By using only the Standard Model it is possible to get an estimation [16] of the neutron EDM at the following level:

$$d_n \approx 2 \cdot 10^{-32}\ e \cdot \text{cm}. \tag{2.1}$$

The **n2EDM** experiment is conceptually following the footsteps of the results obtained by the **nEDM** collaboration. By analysing the data obtained at ILL, Grenoble it was possible to achieve [19] an impressive record of d_n precision:

$$|d_n| < 3.6 \cdot 10^{-26}\ e \cdot \text{cm (95\% CL)}. \tag{2.2}$$

This leaves us with 6 more orders of magnitude to go. The n2EDM experiment aims to cut this number to five, improving [1] the precision tenfold.

What method will be used? Same as in the original nEDM experiment, Ramsey method of the time separated oscillating fields.

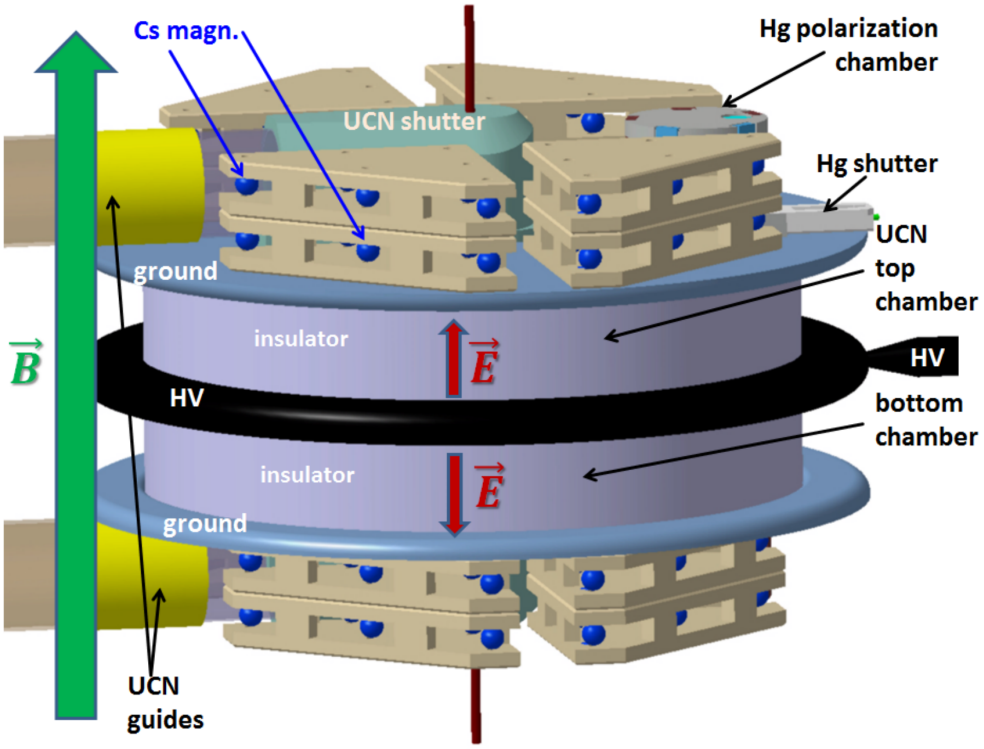

Figure 21: Double chamber design, image taken from [1].

One can see that the precession chamber pictured in the Fig. 21 features fields $\vec{E}$ and $\vec{B}$ that are codirectional in one chamber and contradirectional in another. In both chambers the neutron can be described with the Hamiltonian from Eq. 1.1. Let's take a look at its Larmor precession.

In the case of the **codirectional** fields we can write

$$h\nu_{\uparrow\uparrow} = -2\left(\mu_n B_{\uparrow\uparrow} + d_n E_{\uparrow\uparrow}\right). \tag{2.3}$$

If the fields are **contradirectional** we will get

$$h\nu_{\uparrow\downarrow} = -2\left(\mu_n B_{\uparrow\downarrow} - d_n E_{\uparrow\downarrow}\right). \tag{2.4}$$

By combining Eq. 2.3 and Eq. 2.4 we can express the neutron electric dipole moment d_n through the fields E and B, magnetic moment μ_n and Larmor frequencies $\nu_{\uparrow\uparrow}$ and $\nu_{\uparrow\downarrow}$ as

$$d_n = \frac{h\left(\nu_{\uparrow\downarrow} - \nu_{\uparrow\uparrow}\right) - 2\mu_n\left(B_{\uparrow\uparrow} - B_{\uparrow\downarrow}\right)}{2\left(E_{\uparrow\uparrow} + E_{\uparrow\downarrow}\right)}. \tag{2.5}$$

Why is the double chamber design important? The idea of a double chamber pioneered [2] in 1980. The biggest improvement that it brings is the ability to measure the Larmor frequency for the codirectional and the contradirectional cases **simultaneously**. This feature allows to strongly reduce [1] any time dependent systematic effects.

How does n2EDM look like schematically? You can see it on the Fig. 22.

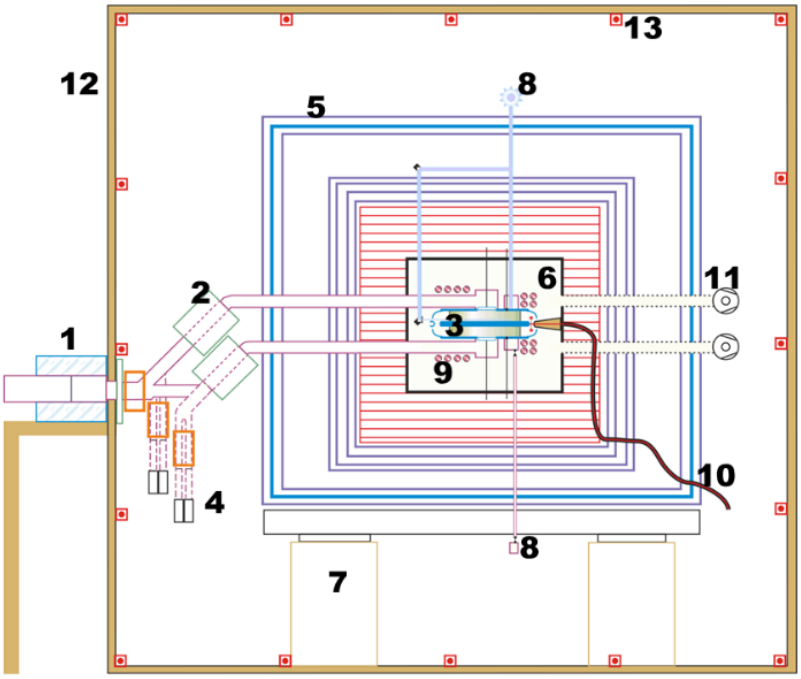

Figure 22: Schema of the experimental setup as portrayed in [1].

It features [1] the following:

1. A 5 T superconductive polarizer magnet to align the spin of UCNs before they enter precession chambers.
2. Switches to control the filling and emptying of the UCN chambers.
3. Two precession chambers, portrayed in details on Fig. 21.
4. Four spin projection detectors — for every chamber we count amount of neutrons with spins up and down.
5. Magnetically shielded room to protect the storage chambers and the vacuum vessel from the external magnetic fields.
6. The vacuum vessel.
7. Four granite pillars supporting an Al plate.
8. The Hg magnetometer to measure the average magnetic fields.
9. The Cs magnetometer to measure the gradients of the magnetic field.
10. A high voltage cable.
11. The molecular pumps generating vacuum in the vacuum vessel.
12. Insulation shell, thermally stabilized by air-conditioning (not shown).
13. Surrounding field compensation (SFC) system is designed to actively minimise the magnetic perturbations of the environment.

Figure 23: Recent photo of the n2EDM experiment shot by Georg Bison.

Chapter 3

The n2EDM DAQ system

In order to measure the neutron EDM and question the theoretical predictions mentioned in the Chapter 1 it is not enough to do a single cycle of the experimental setup from the Chapter 2. Everything comes at a price and pushing the limits of precision is not an exception. The analysis [20] of the previous nEDM experiment was based on data collected between 1998 and 2002, with each data-taking run lasting about 1–2 days. Thus a solid and performant data acquisition and control (DAQ) system is strongly needed.

Why cannot we just reuse the DAQ from nEDM? Apart from an experimental setup containing new modules and equipment there are [5] other reasons for us to consider designing a new generation of the DAQ system:

- Complexity of the codebase (one of the projects consisted of approximately 748 332 LabView VIs) was limiting modifications.
- Inability to test or debug the DAQ system without the complete experimental environment, including the hardware, being connected. This was blocking data acquisition or the regular shift routine.
- No standardisation in connecting various hardware devices to the DAQ, resulting in the code repetition.
- Windows operating system lock-in.

What principles is the n2EDM DAQ built on? The new **n2EDM** DAQ aims to address the main pain points and limitations of the old **nEDM** DAQ by selecting to follow the design ideas listed below:

- **TCP/IP communication**: by relying on the TCP/IP as the transport layer we can guarantee deliverability of messages in the same order as they were sent. By being an industrial standard it also simplifies connection of the new hardware nodes to the system. Not specifically TCP/IP-related bonus is that by optically decoupling our hardware it is possible to automatically provide electrical insulation.

- **SCPI syntax**: all commands should be written following a human-readable specification [27]. This would standardise the environment and allow for the simpler debugging.
- **Script control**: what can be better than a single SCPI command? Only a human-readable repeatable set of commands, providing an ability to program the behaviour of the experimental setup.
- **Modularity**: usage of the small loosely connected independent modules improves robustness and encourages testing. Additionally if the modules can be tested without the presence of each other it also becomes simpler to implement the End-to-End (E2E) testing of the whole system with the aim of being able to arbitrary replace physical equipment with its software-only analogs. This brings us closer to an ability to run the n2EDM experiment in a simulation mode.
- **Linux based**: apart from being free (both as in "free as a speech" and "free as a beer") the development ecosystem of Linux provides much more opportunities compared to the Windows one. Even though the recent release [17] of the Windows Subsystem for Linux made the difference less painful, one might still prefer to run the programs directly on Linux with zero overhead.

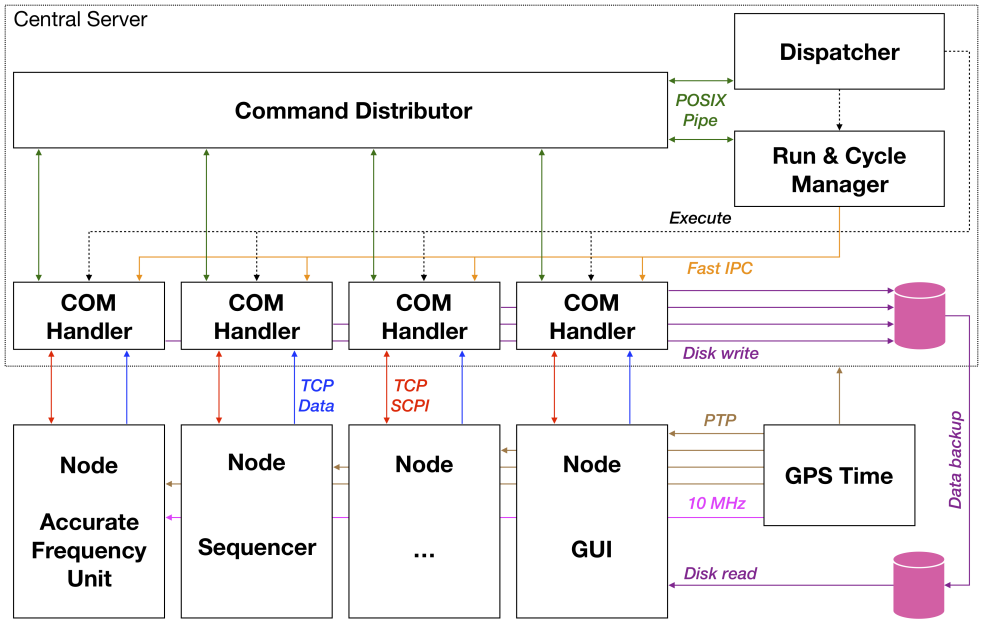

Figure 31: Schematic view of the n2EDM DAQ system. Based on [5].

Let's take a closer look at some of the core components of the n2EDM DAQ system presented on the Fig. 31.

3.1 Command Distributor

https://gitlab.com/n2edm/n2sim

The distributor acts like a spinal cord of the system, combining the functions of the message bus and the modules registry. Every node registers itself with the distributor by providing its name, upstream and downstream FIFOs. For example, the GUI node might provide the following information to the distributor:

- **Module name**: `GUI`
- **Upstream file**: `/n2edm/fifos/_GUI_upstream`
- **Downstream file**: `/n2edm/fifos/_GUI_downstream`

Now another node wants to communicate with the GUI node. To do that it would send `GUI:COMMAND` to the distributor. Distributor would check whether the module with name `GUI` has been registered. If this is the case the message would be forwarded to the appropriate FIFO, in our example that would be `/n2edm/fifos/_GUI_downstream`.

Direct communication with the distributor is possible by writing into a hardcoded path `/n2edm/_n2sim_input`. User can simulate the behaviour described above by executing this command:

```
echo "GUI:COMMAND" >> /n2edm/_n2sim_input
```

3.2 Dispatcher

https://gitlab.com/n2edm/n2dispatcher

The dispatcher provides a generic mechanism for executing commands on the central server. It should be enough to have the distributor and the dispatcher running to boot up every other module located on the same computer. One could imagine the following flow:

1. Content of the repository https://gitlab.com/n2edm/startscripts is cloned into the `/n2edm/startable` folder of the central server.
2. Scripts in this folder are made executable.
3. Distributor and dispatcher are started.
4. User sends `n2dispatcher:startch 'runcyclenumber','' ` to the distributor.
5. Dispatcher executes `/n2edm/startable/runcyclenumber` and starts the run & cycle manager.

Dispatcher is also reachable directly via TCP/IP. A possible scenario could be that the remote node boots up and asks the dispatcher to start a corresponding COM handler. This way this node would become included into the global communication system.

3.3 Run & Cycle Manager

https://gitlab.com/n2edm/runcyclenumber

This node records and distributes to other nodes information from the UCN source about the run and cycle number. This is needed to be able to store and analyse experimental data. Every node that writes binary data to disk uses this information for the distinguishable file names.

3.4 Data Storage

Since it is expected for the n2EDM experiment to generate loads of data it is absolutely necessary to preserve it for the further analysis. Initially data is written to the disk of the central server, however most likely it would not be able to fit the whole volume generated over 5–10 years, thus demanding a secondary larger storage. In order to simplify data management and avoid conflicts of the file versions information from the Run & Cycle Manager is used to transfer the files only after being completed. These static data samples would be distributed further among the partner universities and additional backups. However some nodes, like a GUI node, would need the data accessible in a nearly real-time mode. For that purpose a volatile copy of the most recent files would be provided with the expected delay being below 10 seconds.

3.5 Timing Infrastructure

One of the principal positions [5] of the n2EDM experiment is synchronous equidistant data sampling. We introduce a heartbeat sampling rate of 10 Hz meaning that every node must write its current state to the disk every 1/10 second and every node need to do it at the same absolute time as all other nodes. In other words, we want to be able to describe the joint state of the system 10 times per second. This approach helps to study systematic uncertainties, eases the correction of correlations and additionally enables us to represent the state evolution as a (FFT) spectrum or as an Allan deviation to simplify the data analysis.

This requires all nodes to have their time synchronised between each other. We solve it by using a central GPS controlled grandmaster clock which provides time to the nodes via the precision time protocol (PTP). The accuracy

guaranteed by the PTP should be enough for the most tasks, however some systems, like the Accurate Frequency Unit from the Fig. 31 additionally use the dedicated 10 MHz lines to further improve synchronisation veracity.

3.6 COM Handler

https://gitlab.com/n2edm/n2comhandler

In order to connect the remote nodes to the distributor we need to utilise their TCP/IP endpoint and stitch it together with the corresponding POSIX pipes. COM handler is a smart bridge that does that and additionally provides the following features:

- Auto-registers itself with the distributor.
- Buffers the commands if the node is unavailable.
- Allows to use the response of the node by other components via a generic `REPLYTO` command.
- Handles generation of the header and data files by using information from the run & cycle manager.

As one can see, the COM handler is one of the essential parts of the n2EDM DAQ system and thus it demands a special attention to the design details.

3.7 Sequencer

https://gitlab.com/n2edm/sequencer

In order to program the DAQ system one needs a programming language and an environment that supports it. Sequencer provides the latter and aims to support a set of basic coding blocks that is flexible enough to describe arbitrary behaviour of the individual nodes and system as a whole:

- **Conditionals**: `IF (...) THEN ... ELSE ... ENDIF`
- **Cycles**: `FOR (...) DO ... DONE`
- **Variables (local value)**: `SET variable = expression`
- **Variables (remote value)**: `SET variable = REQUEST(...)`
- **Direct source manipulation**: `(ADD/INSERT/REPLACE/DELETE)LINE`
- **Navigation**: `LABEL "..."` and `GOTO "..."`
- **Lifecycle management**: `SLEEP ...s`, `PAUSE`, `RESUME`

The list above is non-exhaustive but nevertheless gives an impression of how one can control the n2EDM DAQ system. A more detailed explanation of the role and the capabilities of the sequencer can be found in [12].

Chapter 4

Features

Now that we have an understanding about the way the n2EDM DAQ system is expected to operate we can shift the focus of our discussion to the work that was conducted over the course of this thesis. In this chapter we will explain the way they were designed and implemented, additionally highlighting their capabilities as well as the limitations the future developers and/or users might face.

4.1 Implemented the `FOR` loop

Components affected: **Sequencer** (Section 3.7).

Motivation: While the sequencer is able to execute the pseudo-`FOR` loop as shown on the Listing 4.1 we would still prefer to have it implemented as a standalone construction for the sake of simplicity and reducing the mental overhead of the users. Additionally the usage of `GOTO` command is generally considered [8] harmful, leading to the unstructured spaghetti code [6].

```
1 SET i = 0
2 LABEL "FOR_START"
3 IF $i < 5 THEN
4 ...
5 SET i = $i + 1
6 GOTO "FOR_START"
7 ELSE
8 ENDIF
```

Listing 4.1: Implementing `FOR` with `GOTO`.

Requirements: We introduce 3 new commands that the sequencer should be able to handle:

- `FOR (init; test; iterate)` or `FOR ((init; test; iterate))`. The amount of whitespace characters is arbitrary, round brackets are al-

lowed as a part of every argument, but amount of (and) per argument needs to be balanced. Nested FOR blocks are allowed.

  - init: an expression that is being executed once to init the state of the loop variable. It follows the semantic of the SET operator, for example i = 0 is valid.
  - test: an expression that produces a boolean value when evaluated. We run this check on every iteration, including the first one. Semantically similar to the argument of the IF block, so $i < 5 is allowed. If the output is False, skipping the lines until a DONE block is encountered. Otherwise continues execution as usual.
  - iterate: an expression that is used to modify the loop variable. Gets evaluated at the end of every cycle. It borrows the structure of the SET command, usually using the loop variable itself in a following manner: i = $i + 1.

- DO: contrary to the name it does nothing. If encountered, must have a valid FOR statement in a previous line of the sequence.
- DONE: marks the end of the FOR loop. Redirects to the beginning of the loop for the next iteration.

We should be able to replace the Listing 4.1 with the code of the Listing 4.2.

```
1 FOR (i = 0; $i < 5; i = $i + 1)
2 DO
3 ...
4 DONE
```

Listing 4.2: Example of the FOR loop.

Implementation details:

- Contrary to the implementation of the IF/ELSE/ENDIF blocks the FOR loop does not need to be complete all the times. Rather if we encounter FOR we evaluate the FOR.test condition and continue executing or skipping the next lines based on the condition's output until the matching DONE is found.
- Current scope of the sequencer variables is always global, so one might prefer to use different iteration variables for nested loops to prevent confusing results.
- Sequencer supports powerful inline editing/removal of the lines. We do not attempt to prevent cases like REMOVELINE of the FOR line while iterating or GOTO from the loop body. However the sequencer will always try to find the root of the problem and notify the user about it.
- Default strategy for the non-parseable lines is to skip them.

4.2 Improved the REQUEST command

Components affected: **Sequencer** (Section 3.7).

Motivation: while the core functionality of this feature was provided by [12] the implementation contained a few undesired restrictions:

- At any given moment of time only 1 request could have been pending.
- In order to pause the execution while request was in flight, functionality of SLEEP command was used. This would have lead to unexpected results when combining SLEEP, PAUSE, RESUME and REQUEST blocks.
- Without the Feature 4.3 it could have not been tested thoroughly and contained a number of bugs.

Requirements: sequencer needs to support the `SET variable = REQUEST( question, format, timeout, default)` instruction with the semantics as described below:

- `question`, required quoted string starting with `:`. Represents the node name and the command that should return the value that we are interested in. Example: `":HV:OUTPUT:VOLTAGE?"`.
- `format`, optional string (default: `%0`). In case the target node returns multiple comma-separated values this argument in the form of `%n` selects the n-th element starting with 1. Special case `%0` instructs to return the node response as a whole.
- `timeout`, optional integer/double (default: `1`). Maximum amount of seconds the sequencer needs to wait for the answer.
- `default`, optional integer/double (default: `0`). The value used in case the request does not return an answer within the `REQUEST.timeout`.

First step in getting the answer is sending the request itself. Let's discuss how every REQUEST statement should be processed before getting sent to the distributor. `REQUEST(":HV:OUTPUT:VOLTAGE?", %2, 1, 0)` would serve us as an example:

1. We select the `nodeName` from the `REQUEST.question`, in our case it would be the `:HV`, leaving `:OUTPUT:VOLTAGE?` as a `nodeCommand`.
2. We create a quoted `argumentsString` which looks like

   `"sequencerNodeName:RESULT requestId, REQUEST.format"`

   - `sequencerNodeName` is the name with which the sequencer was registered with the distributor, let's assume it is SEQUENCER.
   - `requestId` is a unique positive integer used to distinguish the received answers, we would select 17.

In our case argumentsString would be "SEQUENCER:RESULT 17, %2"

3. We create a final requestCommand of the shape

   nodeName:REPLYTO(argumentsString)nodeCommand

   or for the example that we have started with

   HV:REPLYTO("SEQUENCER:RESULT 17, %2"):OUTPUT:VOLTAGE?

4. The requestCommand is sent to the distributor.

Second step is receiving the answer. To do that, sequencer supports the RESULT command. Following the example we expect to receive a string RESULT 17, remoteValue and execute SET variable = remoteValue. One could notice that the node answer looks like if the argumentsString from above would have been used as a template.

Sequencer supports 2 sources of commands, called external and internal. External commands are the one that arrive from the distributor. Internal commands come from a sequence of commands that the sequencer stores in memory. A command can be loaded into this sequence from the external commands, for example with ADDLINE "...". For this feature we are only interested in a subset of commands from both sides:

- **External**:
  - PAUSE. Pauses the execution of all internal commands until the RESUME is received.
  - RESUME. Resumes the execution of all internal commands if paused by PAUSE.
  - RESTART. New command, removes information about all requests, rewinds the internal sequence to the first line, cancels any blocks introduced by PAUSE or SLEEP. **Preserves** the state of all variables.
  - SET variable = REQUEST(...). Pauses the execution of the internal commands until either an answer is returned or the timeout of the corresponding request was reached, whatever event happens earlier. Sets the variable to the received or default value.
- **Internal**:
  - SLEEP ...s. Pauses the execution of the internal sequence for a given amount of seconds. Is **not** equal to pausing the sequencer with PAUSE.
  - SET variable = REQUEST(...). Same logic as if it would have been executed directly from the external source.

Implementation details:

- The execution of the internal sequence can be paused independently by multiple events:
  1. Both `PAUSE` and reaching the end of the sequence set the internal `isPaused` flag to `True`.
  2. `SLEEP` sets the `isSleeping` flag to `True` and saves the time at which it should be set back to `False`.
  3. At least 1 pending request is in flight.

  The execution continues only in case of none of these blocking conditions being present.
- A somewhat unexpected consequence of sharing the `SET` logic is that the `REQUEST` can be also used as part of the Feature 4.1. Both `FOR.init` and `FOR.iterate` can be `REQUEST`ed, making the following line valid:

```
FOR (i = REQUEST(...); $i < 5; i = REQUEST(...))
```

4.3 Implemented the REPLYTO command

Components affected: **COM Handler** (Section 3.6).

Motivation: in order to create a truly interconnected DAQ system it is not enough to be able to send SCPI commands to our nodes. We also need to be able to use their responses for a feedback loop of any kind. By implementing this feature on the COM handler level it would be possible to return a response of any node.

Requirements: following the contract of the `REQUEST` Feature 4.2 the COM handler needs to support a `REPLYTO` command. We would like to achieve a behaviour as described below:

- Only one `REPLYTO` command can be processed at a time. When it is executed the COM handler should continue accepting commands from the distributor. These commands would be buffered in memory until the pending `REPLYTO` command completes.
- Given the example from the Feature 4.2

  ```
  HV:REPLYTO("SEQUENCER:RESULT 17, %2"):OUTPUT:VOLTAGE?
  ```

  we will send `OUTPUT:VOLTAGE?` to the node and pause the further execution of the SCPI commands.
- Upon executing the `REPLYTO` command the COM handler sets a time window based on the integer `scpiResponseTimeoutMs` value from the config file `conf_n2comhandler.cfg`. We have 3 cases to handle:

  - If nothing came over the SCPI connection with the node during the waiting interval then it means that REPLYTO has timed out. We resume the execution of the SCPI commands from the distributor from the next command, if any.
  - If a complete message that starts with `:` was received then we assume that the node is trying to communicate with other components of the system. Such message is forwarded to the distributor directly as is and does not influence the REPLYTO feature flow.
  - Otherwise if a complete message was received while we are still within the time window of waiting for response then we consider it to be an answer to the REPLYTO being in flight.
  - If a node answer that does not start with `:` was received outside of the REPLYTO timeout the COM handler would skip it and print a warning.

- If a complete SCPI response was received in time then our next task is to parse it. A node might decide to return multiple comma-separated values. We split the message into chunks by using the rules below:
  - Message is scanned from left to right.
  - Splittable parts are separated by commas `,`.
  - Escaped commas `\,` are not separators.
  - Commas that appear inside of a string are not separators.
  - String starts with a quotation mark `"`.
  - String might end with a quotation mark `"`. If there is no closing quotation mark, we consider the string to end at the end of the message. So in `"1,2,3` none of the commas would be a separator.
  - Escaped quotation marks `\"` are not considered quotation marks.
- After splitting the message we need to extract the correct `answerValue` by using the format argument, in our example that would be `%2`
  - `%0` means that no splitting is required and `answerValue` is set to the received message as a whole.
  - `%n` means that `answerValue` is set to the `n`-th part of the split message with count starting with 1. If the message contained no separators then `%0` and `%1` produce the equal output. If the split message contains less than `n` parts we set `answerValue` to an empty unquoted string.

- After extracting the `answerValue` we are ready to send back the answer. Let's assume that `answerValue = 289`. Following our example, the COM handler will need to send `:SEQUENCER:RESULT 17, 289` over FIFO to the distributor.

Implementation details:

- In the *requirements* above one can often see a combination of words **complete message**. Neither TCP/IP nor FIFO files guarantee that the message sent as one would be received as one. All SCPI messages, such as
  - Commands from the distributor.
  - Answers to the distributor.
  - Command to the node.
  - Answers from the node.

  must use a newline `\n` symbol to indicate the ending of the command. Internally we have 2 string buffers: for the commands from the distributor and the answers from the node. Every received message is placed into a corresponding buffer. On every tick we attempt to select one **complete** message by splitting once on the newline symbol `\n` starting from the beginning of the buffer. In case of success this **complete** message is placed in a corresponding sequence and treated as a SCPI command/answer. The SCPI answers buffer is not cleaned on the `:REQUEST` timeout.
- We consider a SCPI response to be an answer to `REPLYTO` if it was received during the timeout window. Even though the default value of `scpiResponseTimeoutMs` being 5000 should be enough for the node to return the answer in time, we cannot guarantee it. Thus a very late answer that comes during the execution of the **next** `REPLYTO` command would be sent as an answer to the distributor.
- Combining both points we might get a very confusing situation:
  1. COM handler executes first `REPLYTO`.
  2. Node sends back `verylonganswer\n`.
  3. The answer does not fit in a single package and the COM handler received only `very` and adds it to the answer buffer.
  4. Due to the network error the connection between the node and the COM handler is broken. First `REPLYTO` times out.
  5. Network connection is reestablished.

6. COM handler executes second `REPLYTO`.
7. Node sends back `shortanswer\n`.
8. Depending on the node logic, 2 situations would be possible:
   - Node tries to send the remaining `longanswer\n` first. COM handler would have an answer buffer `verylonganswer\n` and send it as a result of the second `REPLYTO`. The received message `shortanswer\n` would be appended to the buffer and depending on timing either ignored or sent as a reply to the third `REPLYTO`.
   - Node skips the `longanswer\n` and decides to respond only with `shortanswer\n`. COM handler would have an answer buffer `veryshortanswer\n` and return it as a result of the second `REPLYTO`.

One can see that with a lot of requests and slow nodes that might become an issue, thus the COM handler would print a warning if its queue of the pending SCPI commands has more than one item.

4.4 Implemented the sequencer's TCP/IP interface

Components affected: **Sequencer** (Section 3.7).

Motivation: connecting the sequencer directly to the distributor via the POSIX pipes would require us to implement the logic that handles it twice: in both the sequencer and the COM handler. In order to reduce complexity we would prefer the sequencer to behave as a regular node that uses a COM handler proxy for the communication with other parts of the DAQ system.

Requirements: sequencer needs to act as a TCP/IP server by listening on 2 ports: one for the SCPI commands and one for the data connection. The only connection that would be used is the SCPI connection. Values for the corresponding port numbers should be provided via a `libconfig`-compliant configuration file.

Implementation details: a configuration file `conf_n2comhandler.cfg` of the COM handler would serve us as an example. Following it we introduce `conf_sequencer.cfg` that needs to be present in the same folder as the `sequencer` executable. It must contain the following fields:

- `name`, string — a short human-readable description of the sequencer.
- `moduleName`, string — is required to provide a `sequencerNodeName` in the Feature 4.2. Must be equal to the `moduleName` of the corresponding COM handler config.

- `ipAddr`, string — an interface that the sequencer needs to be listening on, for example `127.0.0.1`. Might be marked obsolete by modifying the sequencer code to listen on all network interfaces. In this case it would be ignored and thus one could completely reuse the `conf_n2comhandler.cfg`.
- `cmdPort`, integer — a port number for the TCP server to listen for the SCPI commands.
- `dataPort`, integer — a port number for the TCP server to accept the data connection. Following the example of the COM handler a default value of 50250 was used. However as it was later found during the implementation of the Feature 4.5 a value outside of the ephemeral port range would be preferred.

We provide an example of the configuration file that uses the default values below on the Listing 4.3:

```
1 name = "sequencer - a n2EDM scheduler for SCPI commands";
2 moduleName = "SEQUENCER";
3 ipAddr = "127.0.0.1";
4 cmdPort = 5025;
5 dataPort = 50250;
```

Listing 4.3: Example of the `conf_sequencer.cfg`.

It is important to notice that in the current setup the sequencer would wait for the TCP connections only on the start of the execution. Any network error or disconnection of the COM handler would cause the sequencer to print a warning and exit. Whether the sequencer needs to handle these cases differently is up for discussion.

4.5 Improved the COM handler's networking

Components affected: **COM handler** (Section 3.6).

Motivation: we live in a world where the network errors are part of the reality. It is unacceptable for the COM handler being one of the most important communication bridges in the DAQ system to shut down in case of the connected node becoming unreachable over the network.

Requirements: instead of a single attempt to connect to the node when being started the COM handler needs to graciously manage the corresponding network connections:

- SCPI and data connections are managed in a same fashion but independently, errors in one of them should not affect the other.
- We try connecting to the node as many times as needed to establish a network link.

- If the COM handler encounters an error in the SCPI or data connection it should properly close it and start reconnecting.
- If the error was raised in the SCPI connection while sending a command to the node, this command should not be lost but rather resent as a first message when a new SCPI connection is established.

Implementation details:

- REPLYTO commands are sent to the node when the SCPI connection becomes available. The deadline for the node response is calculated at the moment of sending the command. When set, the time window does not account for the disconnects:
  1. Waiting time is 5 seconds.
  2. At time T the COM handler sends the SCPI command to the node.
  3. At time $T + 1$ seconds the SCPI link goes down.
  4. At time $T + 5$ seconds the COM handler considers the request to be expired.
  5. At time $T + 6$ seconds the SCPI connection is reestablished.
  6. At time $T + 7$ seconds the node returns the answer but it is not considered valid, even though the waiting time with the **open** SCPI connection was only 2 seconds.
- Persistent attempts of the COM handler to open a data connection to the 127.0.0.1 without the node being reachable might lead to an unexpected outcome — the connection would be established with the COM handler itself! As briefly mentioned in the Feature 4.4 it has to do with the fact that the default data port number used to be in the ephemeral port range [24, 29]. Although such behaviour is consistent with the RFC [21] it is not desired while operating the DAQ system. One is advised to use both the SCPI and the data ports outside of the ephemeral range in order to prevent confusions.

4.6 Improved the SHOWVARIABLES? command

Components affected: **Sequencer** (Section 3.7).

Motivation: with the Features 4.3 and 4.4 implemented it becomes possible to use the sequencer's output by other nodes. One of the things that might be of interest is the snapshot of the variables of the sequencer.

Requirements: the sequencer needs to handle the SHOWVARIABLES? command by sending a single string over the SCPI connection. This message is created with the following rules in mind:

- Any variable forms a string `variableName=variableValue`.
- An additional variable `LINE_EXECUTED_NEXT` is included. Its value is defined as the number of the line in the **internal** sequence (more details in the Feature 4.2) that would be executed afterwards. The count starts with 0.
- All individual chunks are joined by using `|` as a separator.

Implementation details: assuming that the sequencer is registered with the distributor under the name `SEQUENCER` and the distributor's main FIFO file path is `/n2edm/_n2sim_input` the following set of the shell commands:

```
1  echo "SEQUENCER:ADDLINE SET x = 17" >> /n2edm/_n2sim_input
2  echo "SEQUENCER:ADDLINE SET y = 289" >> /n2edm/_n2sim_input
3  echo "SEQUENCER:RESUME" >> /n2edm/_n2sim_input
4  sleep 5
5  echo "SEQUENCER:SHOWVARIABLES?" >> /n2edm/_n2sim_input
```

would cause the sequencer to send back this string over SCPI:

```
LINE_EXECUTED_NEXT=2|x=17.000000|y=289.000000\n
```

One can see that the sequencer has correctly executed 2 `SET` lines and now the `LINE_EXECUTED_NEXT` is pointing to the virtual third line which represents the end of the sequence.

4.7 Improved the SHOWLINES? command

Components affected: **Sequencer** (Section 3.7).

Motivation: similar to the Feature 4.6 we would like to return the information about the lines of the internal sequence over the SCPI connection.

Requirements: we process the lines for `SHOWLINES?` command almost like we process the variables for the `SHOWVARIABLES?` command. There are, however, a few differences to keep in mind:

- Syntax of individual chunks is changed to `lineNumber:lineValue` with `lineNumber` starting from 0.
- `LINE_EXECUTED_NEXT` needs to be added as a virtual `lineNumber`.
- Line chunks are joined by using `|` as a separator.
- When joining the lines all virtual lines need to be in the beginning. Real lines following the virtual lines block need to be presented in the same order as they are kept in the sequencer's memory.

- Contrary to the variable names and values, the `lineValue` might contain `|` and thus it could be escaped. We use a logic that somewhat resembles the one presented in the Feature 4.3:
  - The line needs escaping if it contains at least one qualified `|`.
  - The line content is scanned from left to right .
  - Escaped separators `\|` do not qualify.
  - Symbol `|` that appears inside of a string does not qualify.
  - String starts with a quotation mark `"`.
  - String might end with a quotation mark `"`. If there is no closing quotation mark, we consider the string to end at the end of the line. So in `"text|moretext|` none of the `|` would qualify.
  - Escaped quotation marks `\"` are not considered quotation marks.
- If the line's content needs escaping, we would form the `lineValue` by following these rules:
  1. All double quotes `"` are transformed into `\"`.
  2. Escaped content is placed between two quotation marks `"`.

Implementation details: assuming the same configuration as in the Feature 4.6 we can execute

```
1  echo "SEQUENCER:ADDLINE SET x = 17" >> /n2edm/_n2sim_input
2  echo "SEQUENCER:ADDLINE SET y = 289" >> /n2edm/_n2sim_input
3  echo "SEQUENCER:RESUME" >> /n2edm/_n2sim_input
4  sleep 5
5  echo "SEQUENCER:SHOWLINES?" >> /n2edm/_n2sim_input
```

to order the sequencer to reply with:

```
LINE_EXECUTED_NEXT:2|0:SET x = 17|1:SET y = 289\n
```

As expected, the value of the `LINE_EXECUTED_NEXT` is consistent with the one that can be received as an answer to the `SHOWVARIABLES?` command.

4.8 Implemented the remote magnetometers' proxy

https://gitlab.com/n2edm/remote-magnetometers-proxy

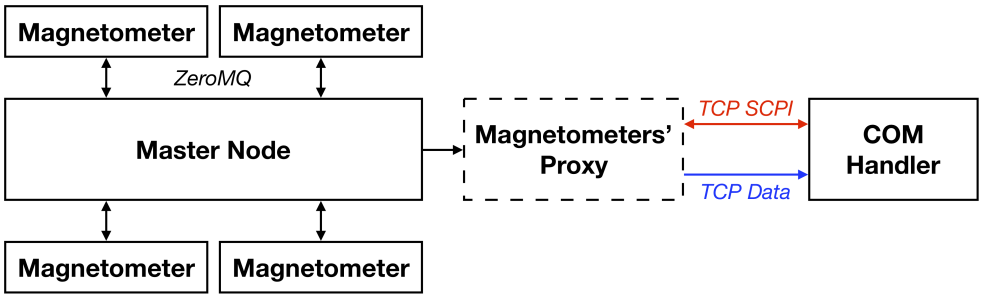

Figure 41: Position of the remote magnetometers' proxy in the n2EDM DAQ system.

Magnetometers' code: https://gitlab.com/n2edm/remote-magnetometers

Overview: the remote magnetometers' proxy, or the **RM-proxy** for brevity, aims to be a missing link between the master node orchestrating a pool of Raspberry Pis with sensor panels and the standard TCP/IP interface of the COM handler. The remote-magnetometers project was already connected to the old LabVIEW **nEDM** DAQ system, however with this adapter it becomes possible to integrate it into the new **n2EDM** DAQ system.

General principles: in order to make the RM-proxy robust and ideally allow it to run without downtime over the whole course of the n2EDM experiment the values below were adopted:

- **Handle errors** — errors happen and the RM-proxy is not an exception. We attempt to graciously process them, for example a problem with the TCP or ZeroMQ link should lead to closing and recreating the corresponding connection. Additionally every handleable issue should generate and save a report message that can be retrieved for an error analysis. Potentially later the n2EDM DAQ system would have a dedicated exception node to process those.
- **Zero data loss** — in an experiment as complex as the n2EDM experiment every single bit of data might improve the precision of the result. Thus it is essential that the data processing and communication are decoupled from each other. Network issues should never lead to the data loss, rather the data chunk should be saved in the memory and a new attempt to send it should be made when a healthy connection is established again.
- **Trust the newer** — the connected COM handler might get stuck or experience other issues that would not lead to it closing the network

connection. The RM-proxy considers every new client to be the healthiest one by properly closing older connections and only allowing a single COM handler to be connected at any given moment of time.

4.8.1 SCPI interface

Motivation: even though at the current stage the RM-proxy is a passive client of the magnetometers' master node we would still like to have a working SCPI connection for the potential future tasks, for example to be able to remotely update the firmware on the magnetometers.

Requirements: by following the SCPI specification [27] we aim to support a single `SYStem:ERRor[:NEXT]?` command which comes in multiple flavours:

- `SYSTem:ERRor?`
- `SYST:ERR?`
- `SYSTem:ERRor:NEXT?`
- `SYST:ERR:NEXT?`

Any of the commands above replies with a message of the following format:

```
code, "event_description;[event_info;]date"
```

where

- `code`, required integer — an error code from [27] that aims to provide the closest definition of the type of the issue occurred, e.g. `-360`.
- `event_description`, required string — a human-readable description of the `code` as specified in [27], e.g. `Communication error`.
- `event_info`, optional string — custom metadata that we might add to precisely describe the reason of raising the error, e.g. `Error in the SCPI connection!`.
- `date`, required string — a `yyyy/mm/dd HH:MM:SS.sss` timestamp [27] of the moment the error message was created.

Internally errors are stored in a FIFO queue with a size limit of 10^5 units. If the queue is full and we attempt to write a new message then the last message in the queue is replaced with a `-350, "Queue overflow;date"` with the `date` following the rules above.

When processing the `SYStem:ERRor[:NEXT]?` command we attempt to pick a first message in the error queue to be used as a SCPI answer. If the queue is empty we return `0, "No error;date"` where the `date` is constructed at the moment of processing the command as it was defined before.

Implementation details: if the network connection is broken when we try sending back `0, "No error;date"` then this message would be placed in the FIFO queue together with the corresponding network error to be resent later.

4.8.2 Data interface

Motivation: in order to store the data collected by the remote magnetometers we need to pack it into the appropriate [5] binary format and send to the COM handler over the TCP data connection.

Requirements: we expect the messages received from the master node over the ZeroMQ connection to look like `node_message; node_message; ...` and to contain as many `node_messages` as there are the remote magnetometers connected. Every `node_message` can either be a `nan` string if the root node was not able to acquire the individual node measurements in time or a space-separated string `value_1 value_2 ...` otherwise, with every `value_*` being a float. The Table 41 dives into more details:

Position	Description	Units
1	Timestamp of the node	seconds
2	Magnetic field x	μT
3	Magnetic field y	μT
4	Magnetic field z	μT
5	Temperature	°C
6	Pressure	mbar
7	Humidity	%
8	Rotational speed x	radians/second
9	Rotational speed y	radians/second
10	Rotational speed z	radians/second
11	Acceleration x	Gs
12	Acceleration y	Gs
13	Acceleration z	Gs

Table 41: Structure of the `node_message`.

Implementation details: we need to process the received message and pack it into the format that the COM handler would understand. Let us take a look at the n2EDM DAQ specification [5]. We see that every separate field in the joint binary data message needs to be presentable as a 64 bit value with the first one always being a timestamp in nanoseconds since 1970, usually in `uint64`. Luckily we can [30] represent the `NaN` value as a `double` thus we will attempt to construct a structure of the following shape:

$$\underbrace{\texttt{timestamp}}_{\texttt{uint64}} \qquad \underbrace{\texttt{value\_*}}_{\texttt{double}\times 13\times\text{number of magnetometers}}$$

We define the `timestamp` as a minimal time among all the `node_messages` being present in a single ZeroMQ message. But one inevitably needs to handle the case of every `node_message` in a package being `nan`, we would refer to such packages as **empty**. The specification for dealing with them is presented below:

1. On the RM-proxy start we wait for the first non-empty package. All empty messages received before that moment would be thrown away during the processing stage.
   a) We use the `timestamps` of individual `node_messages` to calculate and cache the `timestamp` of the binary message that would be later sent to the COM handler.
   b) Repeat the previous step until an empty message is received.
2. On this step the RM-proxy needs to handle an empty message.
   a) Following the Section 3.5 we can calculate a new `timestamp` by using the refresh rate of the master node and a cached value of the preceding package's `timestamp`.
   b) We cache the new `timestamp`.
   c) *Was the previous message of the master node empty?*
      - **Yes** — this empty message is skipped to save the disk space.
      - **No** — we will send the first empty (only a timestamp is included) binary package to the COM handler. The warning is added to the error queue of the Feature 4.8.1.
   d) Repeat until a first non-empty message is received.
3. On a first non-empty message we go to the first step.

4.8.3 Generation of the configs

Motivation: we have 3 projects that share the configuration data: RM-proxy, COM handler and the master node. Apart from them, a configuration for the COM handler requires the specification for every binary field of the data package from the Feature 4.8.2. One can see that the total number of fields to fill $1 + 13 \times N$, where N is the number of magnetometers, quickly becomes unmaintainable by hand and error-prone.

Requirements: to solve these issues we introduce a single source of truth: a root config and a script that treats other configs as the build artefacts. The root config is written in a `libconfig`-compatible format.

Implementation details: Assuming that one is in the root of the RM-proxy's repository we will take a look at the `src/root_config.cfg` on the Listing 4.4 and explain every field.

```
moduleName = "MAGNETOMETERS";
proxyIpAddress = "127.0.0.1";
proxyScpiPort = 5025;
proxyDataPort = 5026;

masterNodeIp = "192.168.0.254";
masterNodeBroadcastPort = 40003;

nodes = (
    {
        number: 1,
        location: "On the table (left)",
        disabled: true
    },
    {
        number: 2,
        location: "On the table (center)",
        disabled: false
    },
    {
        number: 3,
        location: "On the table (right)",
        disabled: false
    },
    {
        number: 4,
        location: "Under the table",
        disabled: false
    }
);
```

Listing 4.4: Root config of the RM-proxy project.

- `moduleName`, required string — a name under which the COM handler would register itself with a distributor.
- `proxyIpAddress`, required string — is the IP address of the RM-proxy by using which the COM handler can connect to it.
- `proxyScpiPort` and `proxyDataPort`, required integers — define on what ports should the RM-proxy listen and the COM handler should try connecting to.
- `masterNodeIp`, required string; `masterNodeBroadcastPort`, required integer — define the IP and broadcast port of the ZeroMQ endpoint of the master node, to which the RM-proxy connects.
- `nodes`, array — describes the properties of the worker nodes. Every object might contain the following fields:
  - `number`, required integer — is used to generate the IP address in the format `192.168.0.number` and the node name `mag_number`.

The node name is included in the header file generated by the COM handler as a column name, e.g. `humidity @ mag_2`.

- `location`, required string — refers to the physical location of the magnetometer. This is included in the header file generated by the COM handler as a column description, e.g. `[%] @ On the table (center)`.
- `disabled`, optional boolean (default: `false`) — allows to mark a single node as excluded from the data collection. Setting this value to `true` would not include the information about this node into any of the generated configs.
- `controlPort`, optional integer (default: `40001`) — the control port of the magnetometer node.
- `queryPort`, optional integer (default: `40002`) — the query port of the magnetometer node.
- `broadcastPort`, optional integer (default: `40003`) — the broadcast port of the magnetometer node.
- `logPort`, optional integer (default: `40004`) — the log port of the magnetometer node.

In order to generate the configs one needs to run a following command from the repository's root:

```
python3 src/generate_configs.py
```

This would read `src/root_config.cfg` and create or overwrite 3 configs:

- `generated/conf_magnetometers_proxy.cfg` is for the RM-proxy.
- `generated/conf_n2comhandler.cfg` is for the COM handler.
- `generated/nodes.txt` is for the master node.

4.9 Improved the SFC system

gitolite@nedm.psi.ch:babySFC

Overview: Surrounding Field Compensation [11, 23] system, or the **SFC**, is an important part of the n2EDM experimental setup. Its purpose is to sustain a magnetic field of a desired magnitude and direction in the measurement area. This goal is achieved by controlling the currents in a set of the magnetic coils [22] in order to respond to the changes in the magnetic fields of the environment. The aim of the Feature 4.9 is similar to the one introduced in the Feature 4.8. SFC needed to be converted into a regular DAQ node

that can be controlled over the SCPI connection and that can feed the COM handler with the experimental data in a compliant [5] way and shape.

4.9.1 SCPI interface

Motivation: we would like to replace an existing GUI control capabilities with a set of the SCPI commands in order to predictably govern the state of execution and the setup parameters of the SFC system.

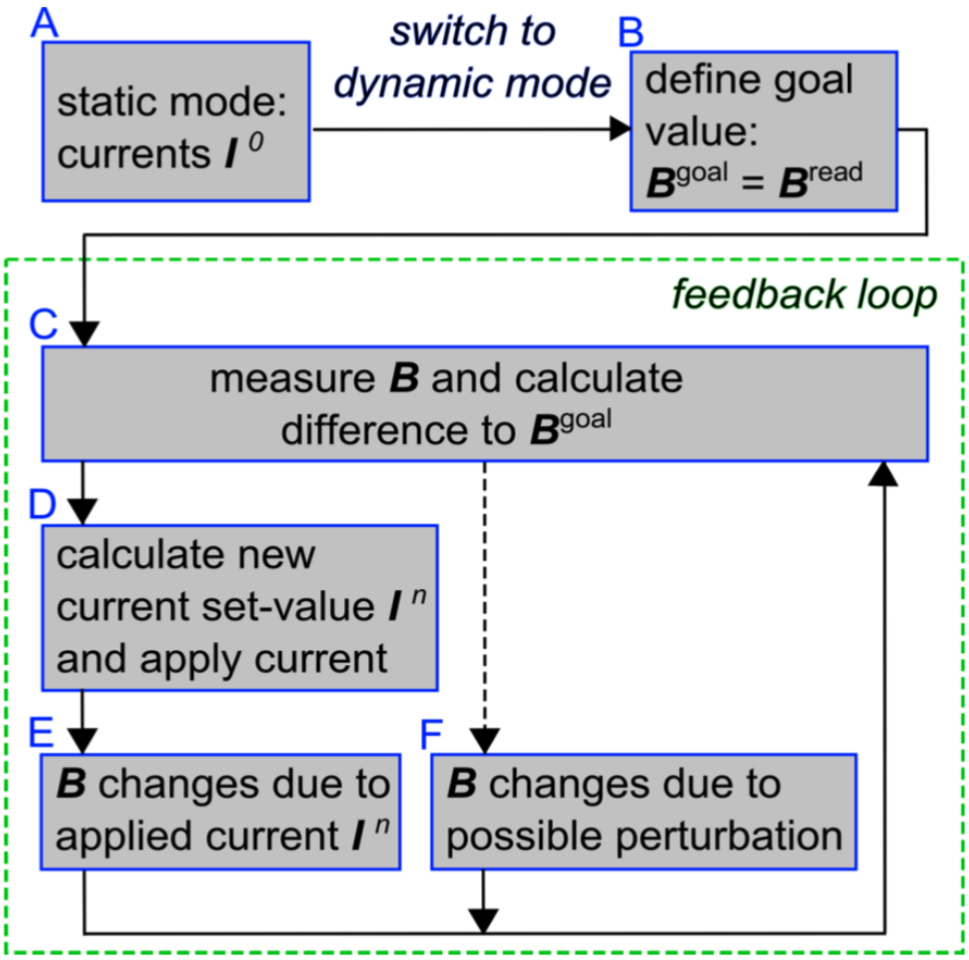

Figure 42: Schematic structure of the SFC lifecycle taken from [11].

Requirements: as described in great details in [11] the SFC system can run in either `STATIC` or `DYNAMIC` mode. While in `STATIC` the system is time-independent and fully characterised with I^0 which is a vector with every individual component defining a ratio of the immediate set current in the corresponding coil to the current at the nominal generated magnetic field.

When switched to the `DYNAMIC` mode the new I^n is recalculated on every iteration based on the difference between the expected and the measured fields. An overview of the SFC behaviour in the different modes can be found on the Figure 42.

We introduce the following 2 commands to control the experimental setup:

- `MODE <STATIC|DYNAMIC>` that switches between the execution modes.
- `COIL index, current` to modify the components of the I^0.
  - `index`, required integer — an index of the corresponding coil. The coil index itself is derived from the coil's position in the config file. Indices start with 1. Any index that is outside the I^0 boundaries is ignored and an error message is printed.
  - `current`, required float — a ratio of the current to be set to the maximal current. However at the moment there are no checks and one can set this value to be greater than 1 if desired.

Implementation details: existing SFC software communicates with the main `babySFC.jl` module via a ZeroMQ connection. This endpoint was kept for the sake of the compatibility. However it is important to notice that these ZeroMQ messages simply contain the **name of the variable** and its new value to be set. Thus one needs to make sure that until the legacy scripts are properly modified to work with the SCPI commands it is necessary to keep the variable name that defines I^0 to be `staticI`.

4.9.2 Data interface

Motivation: in the n2EDM DAQ system the COM handler is responsible for writing the data to the disk for the further analysis thus it is the job of the SFC to bundle it into a binary message and send the package over TCP.

Requirements: we are interested in recording the input and output data of the SFC. The core part of controlling the magnetic field is a set of N_{coils} magnetic coils [22]. Each coil is driven by 3 amplifiers, however this ratio may vary in the future. Every amplifier is connected to a dedicated channel on the Beckhoff EL4134 [4] analog output terminal, referred later to as the **DAC module**. There are $N^{DAC}_{modules} = 3$ DAC modules with $N^{DAC}_{channels} = 4$ channels each. The readout of the magnetic field is performed through a set of the $N_{fluxgates}$ triple-axis fluxgates. In total each of the $3 * N_{fluxgates}$ sensors is connected to a separate channel on the Beckhoff EL3602 [3] analog input terminals or the **ADC modules**. The SFC system operates $N^{ADC}_{modules} = 17$ ADC modules each possessing $N^{ADC}_{channels} = 2$ channels.

Implementation details: let us provide an in-depth analysis of the shape of the data message presented in the Table 42.

Amount of fields	Units	Type	Name
1	ns	`uint64`	`timestamp`
$N^{DAC}_{modules} \times N^{DAC}_{channels}$	V	`double`	`voltage @ dac_channel_*`
$N^{DAC}_{modules} \times N^{DAC}_{channels}$	A	`double`	`current @ dac_channel_*`
$N^{ADC}_{modules} \times N^{ADC}_{channels}$	V	`double`	`voltage @ adc_channel_*`
N_{coils}	%	`double`	`current @ coil_*`

Table 42: Structure of the SFC data message.

- `timestamp` — a UNIX timestamp of the moment when the chunk of data for this message was collected.
- `voltage @ dac_channel_*` — an output voltage in the DAC channel, always 0 if this channel does not have a coil connected.
- `current @ dac_channel_*` — an expected output current of the amplifier connected to the corresponding DAC channel. Always 0 if no connected coil is present.
- `voltage @ adc_channel_*` — a voltage measured in the ADC channel. Either the value read from one of the fluxgate's sensors or noise.
- `current @ coil_*` — a ratio of the immediate current in the coil to the current at a generated nominal field.

4.9.3 Generation of the configs

Motivation: similar to the Feature 4.8.3 we have a need to synchronise the configuration between the COM handler and the SFC system. Additionally the data message described in the Feature 4.9.2 consists of $1 + 2 \times N^{DAC}_{modules} \times N^{DAC}_{channels} + N^{ADC}_{modules} \times N^{ADC}_{channels} + N_{coils} = 62$ fields which makes it unnecessary complicated to be maintained by hand.

Requirements: the only scalable and user-friendly way of addressing the problems mentioned above is to resort to programmatic generation of the configuration files. Following the specification [5] the root config and the configuration file of the COM handler can be consumed by any `libconfig`-compatible parser. For the SFC itself YAML format was selected due to a better support provided by the Julia programming language.

Implementation details: aiming for brevity but without loss of generality we will describe the fields of the reduced (only 1 coil and only 1 fluxgate) root config located in the top level folder of the SFC git repository under the name of `root_config.cfg` and presented on the Listing 4.5.

```
moduleName = "SFC";
sfcIpAddress = "127.0.0.1";
sfcScpiPort = 5125;
sfcDataPort = 5126;

inputDelayCycles = 2;
cycleFrequencyHertz = 200;

dacModulesRange = "1:3";
dacChannelsPerModule = 4;

adcModulesRange = "4:20";
adcChannelsPerModule = 2;
adcInputVoltageRange_in_V = 5;

coils = (
    {
        name: "X",
        amplifiers: (
            {
                dacChannelNumber: 1,
                dacOutputVoltageAtNominalField_in_V: 5,
                amplifierGain_in_AmperPer10Volt: 10
            },
            {
                dacChannelNumber: 5,
                dacOutputVoltageAtNominalField_in_V: 1,
                amplifierGain_in_AmperPer10Volt: 2
            },
            {
                dacChannelNumber: 9,
                dacOutputVoltageAtNominalField_in_V: 0.1,
                amplifierGain_in_AmperPer10Volt: 0.4
            }
        )
    }
);

fluxgates = (
    {
        name: "1",
        location: "top",
        channels: {
            x: 1,
            y: 2,
            z: 3
        }
    }
)
```

Listing 4.5: Root config of the SFC project.

- `moduleName`, required string — an identifier for the COM handler within the n2EDM DAQ system. Is provided to the distributor.

- `sfcIpAddress`, required string — an IP address of the SFC system relative to the COM handler.
- `sfcScpiPort` and `sfcDataPort`, required integers — the ports for the SCPI and data connections, respectively. TCP server of the SFC awaits for the TCP client of the COM handler on them.
- `inputDelayCycles`, required integer — an empirical value of the delay between setting the voltages in the DAC and reading the field change in the ADC.
- `cycleFrequencyHertz`, required integer — a frequency of the SFC feedback and data collection loop.
- `dacModulesRange`, required string — a Julia-inspired [15] range describing the position of the DAC modules in the device chain. Enumeration starts with 1, last value is included. Only a `start:end`, where `start` $<$ `end`, format is supported. It is expected that both the DAC and the ADC modules are installed in homogeneous blocks with the ADC partition immediately following the DAC region. This allows us to address the DAC and ADC channels without the need to specify the module position. For example, a channel 1 in the DAC module 2 can be targeted with `dacChannelNumber` $= 5$ (see below in the `coils` section). Same approach is used for the ADC channels in the `fluxgates` section of the present configuration file. Just like the indexing of the modules themselves, the indexing of the DAC and ADC channels starts with 1.
- `dacChannelsPerModule`, required integer — self-explanatory, is 4 for the Beckhoff EL4134 [4].
- `adcModulesRange`, required string — follows the specification described for the `dacModulesRange` but characterises the ADC modules.
- `adcChannelsPerModule`, required integer — self-explanatory, is 2 for the Beckhoff EL3602 [3].
- `adcInputVoltageRange_in_V`, required integer — defines the input voltage range of all ADC modules.
- `coils`, required array — describes the controlled magnetic coil. Contains the objects with the following set of fields:
  - `name`, required string — a human-readable identifier of the coil, it is included into the generated COM handler's config for the data analysis.
  - `amplifiers`, required array — describes the amplifiers driving this coil. The children have these fields:

        * `dacChannelNumber`, required integer — a number of the DAC channel managing this amplifier. Must be consistent with the specification presented at the `dacModulesRange` description.
        * `dacOutputVoltageAtNominalField_in_V`, required integer — an output voltage of this DAC channel when the corresponding coil is set to generate the nominal field. In the `STATIC` mode this SCPI command from the Feature 4.9.1 would set this voltage value: "`COIL coil_index, 1`".
        * `amplifierGain_in_AmperPer10Volt`, required integer — is used to characterise the connected amplifier.

- `fluxgates`, required array — depicts the fluxgates used to read out the magnetic field. Encloses the objects with the following properties:
    - `name`, required string — a label used to distinguish different fluxgates. Is included into the COM handler's config for the further data analysis.
    - `location`, required string — relates to the physical location of the fluxgate. Is also included in the COM handler's configuration file. `channels`, required object — an object with the keys `x`, `y`, and `z` in a flexible order. Every value of the subfield corresponds to the ADC channel connected to measure the related component of the magnetic field. The channel numbers conform to the specification introduced in the `dacModulesRange` decription.

Executing the command below from the SFC repository root:

```
python3 ./generate_configs.py
```

would read `./root_config.cfg` and create or overwrite 2 configs:

- `generated/conf_sfc.yaml` is for the SFC system.
- `generated/conf_n2comhandler.cfg` is for the COM handler.

Chapter 5

Conclusions

Looking back it is possible to say that the initial task of improving the ease of use and the connectivity of the n2EDM DAQ system was successfully accomplished. The work conducted over the course of this thesis allows both the users and the future developers to be more efficient when operating or building the infrastructure for the n2EDM experiment.

Components affected:

- **Sequencer** (Section 3.7)
  - Added `FOR` loop abstraction (Feature 4.1) simplifies the execution of the repeating code blocks.
  - Better `REQUEST` command (Feature 4.2) opens the doors to more predictable handling of the asynchronous patterns.
  - Standard TCP/IP interface (Feature 4.4) brings all the features that the integration with the COM handler supports.
  - Improved `SHOWVARIABLES?` command (Feature 4.6) sending the result over the SCPI connection makes it possible for the other nodes to know the internal state of the sequencer.
  - Enhanced `SHOWLINES?` command (Feature 4.7) allows the precise remote SCPI control of the sequencer's execution.
- **COM Handler** (Section 3.6)
  - Implemented `REPLYTO` command (Feature 4.3) enables the COM handler to respond with the result of executing the SCPI command by the node regardless of the node type.
  - Revised approach to the TCP/IP networking (Feature 4.5) makes the COM handler suitable for operating in the real life conditions without unintentional downtime.

- **Remote magnetometers' proxy** (Section 4.8)
  - Generic SCPI interface (Feature 4.8.1) powers the specification-compliant [27] error reporting.
  - Data interface (Feature 4.8.2) integrates the pool of the remote magnetometers into the n2EDM DAQ system.
  - Configs' generator (Feature 4.8.3) removes the hurdle to manage the shared setup between the RM-proxy, the master node and the COM handler.
- **SFC system** (Section 4.9)
  - Add information

Potential improvements:

- **Sequencer** (Section 3.7)
  - Add the `BREAK` and `CONTINUE` commands to work with `FOR`.
  - Copy the error-resistant networking logic from the COM handler.
  - Adopt the standard error reporting as in the RM-proxy.
  - Allow for the non-blocking `REQUEST`s.
- **COM Handler** (Section 3.6)
  - Adopt the standard error reporting as in the RM-proxy.
  - Graciously handle potential errors in the POSIX pipes.
- **Remote magnetometers' proxy** (Section 4.8)
  - Extend the SCPI interface to enable the direct control over the remote magnetometers.
  - Automate the distribution of the generated configs.
- **SFC system** (Section 4.9)
  - Add information

Bibliography

[1] C. Abel, N. J. Ayres, G. Ban, G. Bison, K. Bodek, V. Bondar, E. Chanel, P. J. Chiu, B. Clement, C. Crawford, M. Daum, S. Emmenegger, P. Flaux, L. Ferraris-Bouchez, W. C. Griffith, Z. D. Grujić, P. G. Harris, W. Heil, N. Hild, K. Kirch, P. A. Koss, A. Kozela, J. Krempel, B. Lauss, T. Lefort, Y. Lemière, A. Leredde, P. Mohanmurthy, O. Naviliat-Cuncic, D. Pais, F. M. Piegsa, G. Pignol, M. Rawlik, D. Rebreyend, D. Ries, S. Roccia, K. Ross, D. Rozpedzik, P. Schmidt-Wellenburg, A. Schnabel, N. Severijns, J. Thorne, R. Virot, J. Voigt, A. Weis, E. Wursten, J. Zejma, and G. Zsigmond. The n2EDM experiment at the Paul Scherrer Institute. *Nuclear Physics A*, 341(2):269–283, nov 2018. URL: http://arxiv.org/abs/1811.02340, arXiv:1811.02340.

[2] I.S. Altarev, Yu.V. Borisov, A.B. Brandin, A.I. Egorov, V.F. Ezhov, S.N. Ivanov, V.M. Lobashov, V.A. Nazarenko, G.D. Porsev, V.L. Ryabov, A.P. Serebrov, and R.R. Taldaev. A search for the electric dipole moment of the neutron using ultracold neutrons. *Nuclear Physics A*, 341(2):269–283, jun 1980. URL: https://linkinghub.elsevier.com/retrieve/pii/0375947480903139, doi:10.1016/0375-9474(80)90313-9.

[3] Beckhoff Automation GmbH & Co. KG. Documentation for EL36xx Analog Input Terminals (24bit), 2019. URL: https://download.beckhoff.com/download/document/io/ethercat-terminals/el36xxen.pdf.

[4] Beckhoff Automation GmbH & Co. KG. Documentation for EL41xx Analog Output Terminals (16bit), 2019. URL: https://download.beckhoff.com/download/document/io/ethercat-terminals/el41xxen.pdf.

[5] Georg Bison, Jochen Krempel, Dieter Ries, Romain Virot, and Jacek Zejma. N2EDMDAQTDR — second neutron electric dipole moment ex-

periment data acquisition technical design report v0.9. Technical report, 2018.

[6] David Cram and Paul Hedley. Pronouns and procedural meaning: The relevance of spaghetti code and paranoid delusion. *Oxford University Working Papers in Linguistics, Philology and Phonetics*, 10:187–210, 2005. URL: https://www.ling-phil.ox.ac.uk/files/owp2005.pdf#page=187.

[7] Alvaro de Rújula and Rolf Landua. Antimatter Questions & Answers, 2001. URL: https://archive.ph/20080421220420/http://livefromcern.web.cern.ch/livefromcern/antimatter/FAQ1.html.

[8] Edsger Wybe Dijkstra. Go-to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968. URL: http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD215.PDF.

[9] Dirk Dubbers and Michael G. Schmidt. The neutron and its role in cosmology and particle physics. *Reviews of Modern Physics*, 83(4), 2011. doi:10.1103/RevModPhys.83.1111.

[10] Enrico Fermi. Motion of neutrons in hydrogenous substances. *Ricerca Scientifica*, 7(2):13–52, 1936.

[11] Beatrice Franke. *Investigations of the internal and external magnetic fields of the neutron electric dipole moment experiment at the Paul Scherrer Institute*. PhD thesis, ETH Zürich, 2013.

[12] Elsa Germann. Software development for the Supervisory Control and Data Acquisition system of the n2EDM experiment. *Master Thesis*, ETH Zürich, 2019.

[13] R. Golub and Steve K. Lamoreaux. Neutron electric-dipole moment, ultracold neutrons and polarized 3He. *Physics Reports*, 237(1):1–62, 1994. doi:10.1016/0370-1573(94)90084-1.

[14] R. Golub and J. M. Pendlebury. The electric dipole moment of the neutron. *Contemporary Physics*, 13(6):519–558, nov 1972. URL: http://www.tandfonline.com/doi/abs/10.1080/00107517208228016, doi:10.1080/00107517208228016.

[15] Julia language Community. Documentation for Base.UnitRange. URL: https://docs.julialang.org/en/v0.7/base/collections/#Base.UnitRange.

[16] I. B. Khriplovich and A. R. Zhitnitsky. What is the value of the neutron electric dipole moment in the Kobayashi-Maskawa model? *Physics Letters B*, 109(6):490–492, 1982. doi:10.1016/0370-2693(82)91121-2.

[17] Craig Loewen. Announcing WSL 2, 2019. URL: https://devblogs.microsoft.com/commandline/announcing-wsl-2/.

[18] E. Noether. Invariante Variationsprobleme. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1918:235–257, 1918. URL: http://eudml.org/doc/59024.

[19] J. M. Pendlebury, S. Afach, N. J. Ayres, C. A. Baker, G. Ban, G. Bison, K. Bodek, M. Burghoff, P. Geltenbort, K. Green, W. C. Griffith, M. van der Grinten, Z. D. Grujić, P. G. Harris, V. Hélaine, P. Iaydjiev, S. N. Ivanov, M. Kasprzak, Y. Kermaidic, K. Kirch, H.-C. Koch, S. Komposch, A. Kozela, J. Krempel, B. Lauss, T. Lefort, Y. Lemière, D. J. R. May, M. Musgrave, O. Naviliat-Cuncic, F. M. Piegsa, G. Pignol, P. N. Prashanth, G. Quéméner, M. Rawlik, D. Rebreyend, J. D. Richardson, D. Ries, S. Roccia, D. Rozpedzik, A. Schnabel, P. Schmidt-Wellenburg, N. Severijns, D. Shiers, J. A. Thorne, A. Weis, O. J. Winston, E. Wursten, J. Zejma, and G. Zsigmond. Revised experimental upper limit on the electric dipole moment of the neutron. *Physical Review D*, 92(9):092003, nov 2015. URL: https://link.aps.org/doi/10.1103/PhysRevD.92.092003, arXiv:1509.04411, doi:10.1103/PhysRevD.92.092003.

[20] J. M. Pendlebury, W. Heil, Yu. Sobolev, P. G. Harris, J. D. Richardson, R. J. Baskin, D. D. Doyle, P. Geltenbort, K. Green, M. G. D. van der Grinten, P. S. Iaydjiev, S. N. Ivanov, D. J. R. May, and K. F. Smith. Geometric-phase-induced false electric dipole moment signals for particles in traps. *Physical Review A*, 70(3):032102, sep 2004. URL: https://link.aps.org/doi/10.1103/PhysRevA.70.032102, doi:10.1103/PhysRevA.70.032102.

[21] Jon Postel. Transmission Control Protocol. Technical report, 1981. URL: http://www.rfc-editor.org/rfc/rfc793.txt.

[22] M. Rawlik, A. Eggenberger, J. Krempel, C. Crawford, K. Kirch, F. M. Piegsa, and G. Quéméner. A simple method of coil design. *American Journal of Physics*, 86(8):602–608, aug 2018. URL: http://aapt.scitation.org/doi/10.1119/1.5042244, arXiv:1709.04681, doi:10.1119/1.5042244.

[23] Michał Rawlik. *Active magnetic shielding and axion-dark-matter search*. PhD thesis, ETH Zürich, 2018.

[24] William Richard Stevens. *UNIX Network Programming: Networking APIs: Sockets and XTI*, volume 1. Prentice Hall, jan 1998.

[25] D. Sakharov. Violation of *CP* invariance, *C* asymmetry, and baryon asymmetry of the universe. *Soviet Physics - Uspekhi*, 34(5):392–393, 1991. doi:10.1070/PU1991v034n05ABEH002497.

[26] Julian Schwinger. The Theory of Quantized Fields. I. *Physical Review*, 82(6):914–927, jun 1951. URL: https://link.aps.org/doi/10.1103/PhysRev.82.914, doi:10.1103/PhysRev.82.914.

[27] SCPI Consortium. *Standard Commands for Programmable Instruments (SCPI)*, volume 1. 1999. URL: http://www.ivifoundation.org/docs/SCPI-99.PDF.

[28] M. Tanabashi, K. Hagiwara, K. Hikasa, K. Nakamura, Y. Sumino, F. Takahashi, J. Tanaka, K. Agashe, G. Aielli, C. Amsler, M. Antonelli, D. M. Asner, H. Baer, Sw. Banerjee, R. M. Barnett, T. Basaglia, C. W. Bauer, J. J. Beatty, V. I. Belousov, J. Beringer, S. Bethke, A. Bettini, H. Bichsel, O. Biebel, K. M. Black, E. Blucher, O. Buchmuller, V. Burkert, M. A. Bychkov, R. N. Cahn, M. Carena, A. Ceccucci, A. Cerri, D. Chakraborty, M.-C. Chen, R. S. Chivukula, G. Cowan, O. Dahl, G. D'Ambrosio, T. Damour, D. de Florian, A. de Gouvêa, T. DeGrand, P. de Jong, G. Dissertori, B. A. Dobrescu, M. D'Onofrio, M. Doser, M. Drees, H. K. Dreiner, D. A. Dwyer, P. Eerola, S. Eidelman, J. Ellis, J. Erler, V. V. Ezhela, W. Fetscher, B. D. Fields, R. Firestone, B. Foster, A. Freitas, H. Gallagher, L. Garren, H.-J. Gerber, G. Gerbier, T. Gershon, Y. Gershtein, T. Gherghetta, A. A. Godizov, M. Goodman, C. Grab, A. V. Gritsan, C. Grojean, D. E. Groom, M. Grünewald, A. Gurtu, T. Gutsche, H. E. Haber, C. Hanhart, S. Hashimoto, Y. Hayato, K. G. Hayes, A. Hebecker, S. Heinemeyer, B. Heltsley, J. J. Hernández-Rey, J. Hisano, A. Höcker, J. Holder, A. Holtkamp, T. Hyodo, K. D. Irwin, K. F. Johnson, M. Kado, M. Karliner, U. F. Katz, S. R. Klein, E. Klempt, R. V. Kowalewski, F. Krauss, M. Kreps, B. Krusche, Yu. V. Kuyanov, Y. Kwon, O. Lahav, J. Laiho, J. Lesgourgues, A. Liddle, Z. Ligeti, C.-J. Lin, C. Lippmann, T. M. Liss, L. Littenberg, K. S. Lugovsky, S. B. Lugovsky, A. Lusiani, Y. Makida, F. Maltoni, T. Mannel, A. V. Manohar, W. J. Marciano, A. D. Martin, A. Masoni, J. Matthews, U.-G. Meißner, D. Milstead, R. E. Mitchell, K. Mönig, P. Molaro, F. Moortgat, M. Moskovic, H. Murayama, M. Narain, P. Nason, S. Navas, M. Neubert, P. Nevski, Y. Nir, K. A. Olive, S. Pagan Griso, J. Parsons, C. Patrignani, J. A. Peacock, M. Pennington, S. T. Petcov, V. A. Petrov, E. Pianori, A. Piepke, A. Pomarol, A. Quadt, J. Rademacker, G. Raffelt, B. N. Ratcliff, P. Richardson, A. Ringwald, S. Roesler, S. Rolli, A. Romaniouk, L. J. Rosenberg, J. L. Rosner, G. Rybka, R. A. Ryutin, C. T. Sachrajda, Y. Sakai, G. P. Salam,

S. Sarkar, F. Sauli, O. Schneider, K. Scholberg, A. J. Schwartz, D. Scott, V. Sharma, S. R. Sharpe, T. Shutt, M. Silari, T. Sjöstrand, P. Skands, T. Skwarnicki, J. G. Smith, G. F. Smoot, S. Spanier, H. Spieler, C. Spiering, A. Stahl, S. L. Stone, T. Sumiyoshi, M. J. Syphers, K. Terashi, J. Terning, U. Thoma, R. S. Thorne, L. Tiator, M. Titov, N. P. Tkachenko, N. A. Törnqvist, D. R. Tovey, G. Valencia, R. Van de Water, N. Varelas, G. Venanzoni, L. Verde, M. G. Vincter, P. Vogel, A. Vogt, S. P. Wakely, W. Walkowiak, C. W. Walter, D. Wands, D. R. Ward, M. O. Wascko, G. Weiglein, D. H. Weinberg, E. J. Weinberg, M. White, L. R. Wiencke, S. Willocq, C. G. Wohl, J. Womersley, C. L. Woody, R. L. Workman, W.-M. Yao, G. P. Zeller, O. V. Zenin, R.-Y. Zhu, S.-L. Zhu, F. Zimmermann, P. A. Zyla, J. Anderson, L. Fuller, V. S. Lugovsky, and P. Schaffner. Review of Particle Physics. *Physical Review D*, 98(3):030001, aug 2018. URL: https://link.aps.org/doi/10.1103/PhysRevD.98.030001, doi:10.1103/PhysRevD.98.030001.

[29] Joe Touch, Eliot Lear, Allison Mankin, Markku Kojo, Kumiko Ono, Martin Stiemerling, Lars Eggert, Alexey Melnikov, Wes Eddy, Alexander Zimmermann, Brian Trammell, Jana Iyengar, Michael Tuexen, Eddie Kohler, and Yoshifumi Nishida. IANA port number assignments, 2019. URL: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml.

[30] Working Group for Floating-Point Arithmetic. *IEEE 754-1985 - IEEE Standard for Binary Floating-Point Arithmetic*. Institute of Electrical and Electronics Engineers, New York, 1985. URL: https://standards.ieee.org/standard/754-1985.html.

[31] Ya. B. Zeldovich. Storage of cold neutrons. *JETP*, 36(6):1952–1953, 1959.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.