Developing a New Field Test System for Solar Modules

BSc Thesis

EE3L1: Bachelor Graduation Project

Justen van Koppen,
Wessel Oosterkamp

Delft University of Technology

TU Delft

Developing a New Field Test System for Solar Modules

BSc Thesis

by

Justen van Koppen,
Wessel Oosterkamp

Group Supervisor:	Dr. M.R. Vogt
Co-Supervisor:	Dr. T. Silverman & Dr. J.C. Ortiz Lizcano
Project Duration:	April, 2026 - June, 2026
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover: PV monitoring station at the TU Delft [1]. Enlarged with ChatGPT.

TU Delft PVMD Delft University of Technology

Abstract

In recent years, solar modules have gained a significant market share in the electrical energy markets. At TU Delft, researchers develop new solar cells that need to be tested at an outdoor test field. The current test field does not meet the varying solar module sizes and electrical specifications. Therefore, a new testing setup had to be developed.

The new setup comprises a server computer, which runs a database that stores the measurements. To measure the modules, the Open-source Photovoltaic Electrical Tool is used. The microcontroller on this device needed to be adjusted to monitor relevant temperatures.

This thesis describes the development of the PostgreSQL database. The database can synchronise weather measurements from an existing database, keep track of modules that were used, and save point and curve measurements of a solar module. It can do this live and retroactively. The supervisor script, developed using Python, is able to schedule the measurements and store the results in a CSV file. A log file is used to store information about errors, and if any errors occur during the operation, the administrator will be notified via email. Lastly, the firmware of the measurement tool was adjusted to monitor the temperature of a power dissipation device and a solar module. If the temperature of the power dissipation device exceeds the configured limit, the output of the solar module will be disabled.

The end result enables users to access the database remotely and collect data about their solar module. The new system has been validated by a set of pass/fail tests conducted on a separate laptop. The final server for the outdoor testing facility, which runs the program, has yet to be deployed for researchers to use.

Preface

This thesis is about a field test system for solar modules. The field test is developed for TU Delft and its specific infrastructure. This thesis contains information about how the software group developed the measurement setup.

Since the start of our bachelor's, we have been motivated by green technologies. That's why when we saw this project, we were quite certain this one was the one for us. Justen wants to do a Master's in Computer & Embedded Systems Engineering, so he was glad he could develop some firmware. Wessel wants to do Electrical Power Engineering, so the entire project was a fit for that.

What we really liked about this project is that we are really contributing to something. This project was a real engineering project where we had to develop a product that is meant to be used afterwards, which people would find useful. This is unlike many other projects that are done and never to be touched again. We sincerely hope that the system that has been developed delivers on the expectations of the thesis proposer and that it will be used for many years to come.

So we would like to thank all our supervisors. We would like to thank Malte Vogt for supervising the project. We would like to thank Tim Silverman for guiding us with the project and always being open to discussion. We would also like to thank Juan Camilo Ortiz Lizcano for helping us with all the equipment, orders, and basically anything we needed. Lastly, we would like to thank Olivier Meijer and Rein van der Velden for studying together for the past 3 years and for doing this last project together.

Thank you, bedankt, and tige tank!

Justen van Koppen,
Wessel Oosterkamp
Delft, June 2026

Contents

1 Introduction

1.1. State-of-the-art analysis

Since the Paris Accords in 2015, the world has tried to reduce its greenhouse gas emissions. Global greenhouse gas emissions have increased by 30% since 2005 [2]. Nonetheless, renewable sources have gained a 28% market share in electricity generation worldwide [3]. In some regions, emissions have already started to drop. In the EU, for example, emissions have dropped by 33.9% compared to 1990 levels [2].

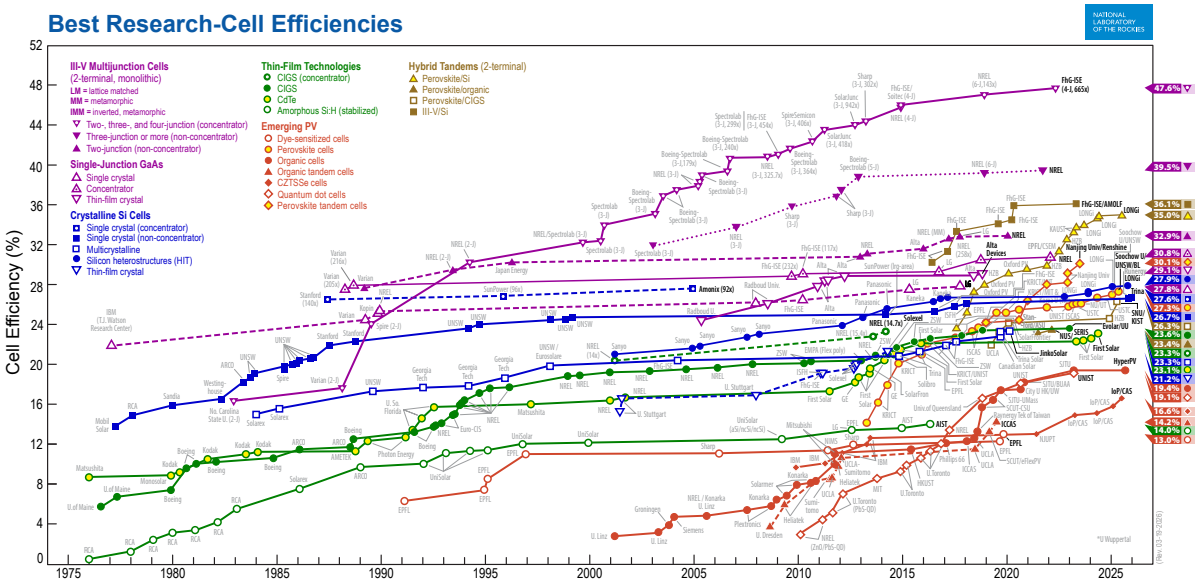

Figure 1.1: Development of the best research cell efficiency from 1976 to today under standard testing conditions.[4]

In recent years, photovoltaic (PV) modules have gained market share in electrical energy markets, reaching 12.8% globally in 2022 [3]. This is due to their low production costs and ever-increasing efficiency (Figure 1.1). The cost of solar module production has dropped to around 0.125 C/W_p for the mainstream class [5]. This reduction in price has increased the total amount of module manufacturing capacity to over 500 GW in 2022 (Figure 1.2). This means that many modules are made and need to be tested after production.

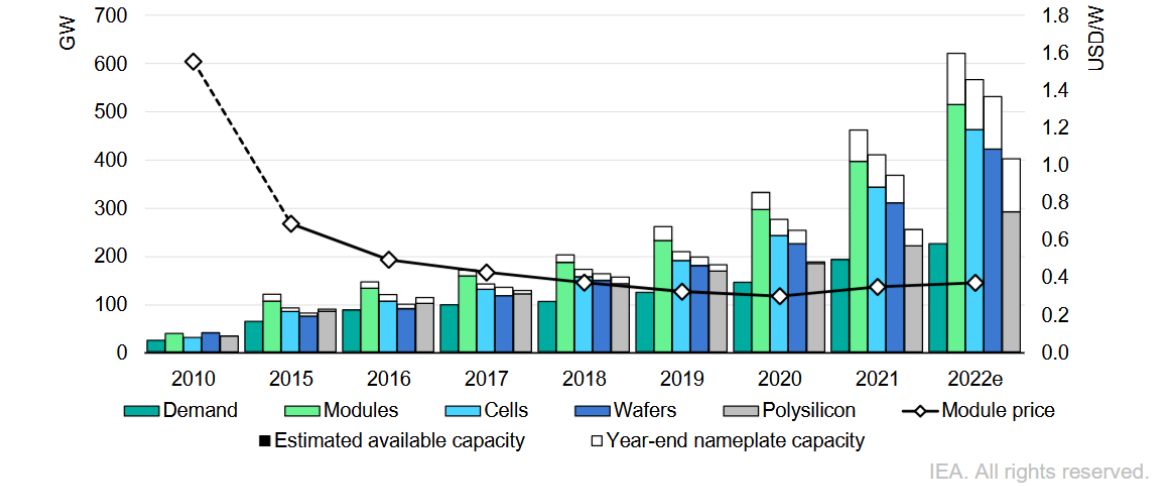

Figure 1.2: Development of global PV manufacturing capacity and average module selling price, 2010-2022 [6]

After manufacturing, a controlled environment test is used. This is done by illuminating the solar module and measuring its cell temperature, output current, and voltage under standard testing conditions [7]. However, this does not give the most accurate representation of real-world performance. The performance of a solar module also depends on wind speed and outside temperature. A field test incorporates these variables into the system and gives real-world performance. This gives live monitoring of the system. Future students and researchers at TU Delft can use the field test to evaluate the performance of their innovative solar modules in an outdoor test.

1.2. Problem definition

1.2.1. Situation assessment

On the TU Delft campus, there is an outdoor testing ground for testing PV modules, which can be seen in Figure 1.3. Cables from the solar modules go inside the building where the measurement instruments are installed. This testing ground makes use of sensors to measure factors such as light intensity, temperature, and wind speed to test the performance of PV modules in different conditions. However, the problem is that this setup does not comply with the use case of the testing ground. This test setup is suitable for larger, high-power PV modules within a small range of specifications. However, PV modules can come in a large variety of voltage and current specifications depending on their material, size and architecture.

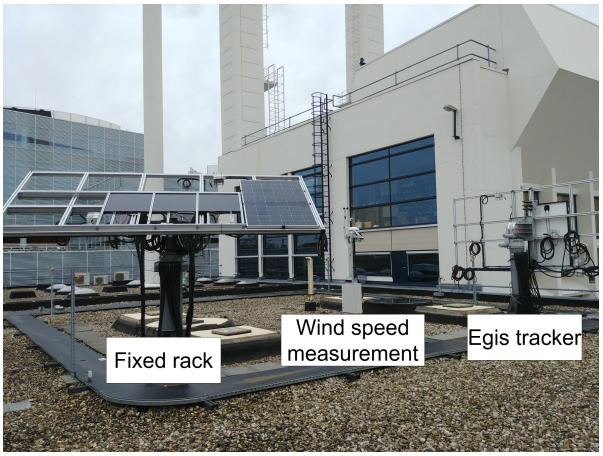

Figure 1.3: Current outdoor PV measurement setup at the TU Delft [1]

As a research facility, TU Delft tests many different types of modules. These modules can range from very small modules with an area of a few square centimetres and an output voltage of less than 1 V, to moderately sized modules of 60 cm by 60 cm, with an output voltage of 100 V. To comply with this use case, where modules may produce a total of 800 W, a new system is introduced in this report.
This system aims to supply TU Delft with the capabilities needed to test PV modules with a much larger range of specifications. The performance data, in the form of output voltage and current, should be integrated with sensor data from the testing ground to provide a clear image of the performance of PV modules under different circumstances.
The main use case of this system will be the testing of PV modules developed by students and researchers. This means the data provided by the system should be made easily accessible for authenticated users.

1.2.2. Scoping Analysis

This bachelor's graduation project (BAP) will focus on the development of two subsystems that will improve the current situation. These subsystems are:

- A benchtop package for at least two PV modules suited for various voltage and current ranges.
- A software program with documentation for storing and processing data from the measurement setup in a database.

For the design of the PV measurement tool, the team will make use of an open-source solution. The Open-source Photovoltaic Electrical Tool (OPET) provides a hardware and software solution for measurements of PV devices under natural or artificial sunlight, with a primary use case for research and development [8],[9],[10]. This was developed by the National Laboratory of the Rockies, which is a laboratory of the U.S. Department of Energy, Office of Critical Minerals and Energy Innovation. Their mission is to have a world with affordable and secure energy [11].

Their hardware module claims to be capable of measuring between 1 V and 100 V with an inaccuracy of <0.1%. The low current version can measure between 1.1 mA and 340 mA, while the high current version can measure between 50 mA and 15 A [8]. Additionally, firmware for these modules and a Python library to communicate with these modules have already been created [9],[10]. The team can use the provided information as a starting point for their setup, as it gives a wider measuring range for the various PV modules. In addition, it provides a foundation for setting up the measurements that can be stored in a database.

1.2.3. Bounding Analysis

As discussed previously, an existing setup is used as a starting point during this project. The goal is to improve this existing system by extending the measurement range and developing software to store and process the gathered data, not to develop a new system from scratch. The complete BAP group consists of four undergraduate students, each having almost 3 years of experience studying electrical

engineering. The product needs to be able to test varying sizes of PV modules. To prove the extended power detection range of the new system, three PCBs should be made, which are able to measure the following power ratings: 800 Watts, 300 Watts, and lower than 10 Watts.

This thesis focuses on the software subgroup consisting of two undergraduate students, who have 10 weeks to create a setup that schedules the measurements and stores them in a database. During this process, familiarity has to be gained with SQL language. Also, the OPET supervisor's original software must be examined and understood so that modifications can be made. Firmware support for additional temperature measurement devices has to be made and added to the OPET firmware.

At the end of the project, a server that collects and stores the OPET measurements data must be delivered. The database running on the server must be accessible remotely. The OPET measurements must have an additional feature to measure the PV cell temperature. Additionally, the OPET firmware must be updated to monitor the temperature of the new power dissipation device.

The other subgroup focuses on the hardware implementation. They design a device that will be able to dissipate up to 800 W. For this design, the firmware of the OPET must be adjusted such that it will be able to monitor the temperature of this new device.

1.3. Use of Generative AI

We did use generative AI (mainly ChatGPT) for this thesis. We are fully aware that we remain responsible even though we have used generative AI. We have used AI for debugging when errors were reported by the compiler, which we could not fix ourselves easily. We mostly used AI to help us get past errors. Also, for the firmware, we have asked it to check our code to see if it would work correctly, because at many moments we could not try it out, because the hardware was not available yet. The AI is able to point out mistakes, such as typos, but also logical errors that would otherwise result in various bugs.

We have only asked generative AI to write a complete Python script for some plotting functions. For example, in `live_data.py`, we have asked AI to create a function that plots the voltage and current simultaneously in a single figure. Creating a nice-looking live plot might take us some time, but the AI is able to do it much faster. As it is not part of the functioning system and is mainly to visualise the results, we thought it was okay to use it. Scripts that are mostly created by AI have a disclaimer in the first code line that states that it was generated with AI.

Additionally, Grammarly has been used as a spelling and grammar checker for writing this thesis. Although Grammarly recently has new AI writing functionalities, these were not used in this thesis.

Finally, ChatGPT was used for the cover image of the thesis. There was no picture available of the test field in the correct aspect ratio. Therefore, ChatGPT was consulted to enlarge the picture to make sure it fits nicely on the cover page.

2
Programme of Requirements

In this section, the requirements of the system that is to be developed are discussed. First, the stakeholders of this project and their needs are defined. Then the functions that the system needs to be able to perform are defined, and lastly, the specific system requirements are defined.

2.1. Stakeholders

The stakeholders related to the project should be taken into account in order to set the requirements. For this project, two main stakeholders are identified, namely: Researchers/students, the PVMD group. They are the stakeholders who will directly work with the system. Additionally, there are two secondary stakeholders. TU Delft provides funding and the location, while the PV community might benefit from better research. Each stakeholder is discussed in more detail below.

- **Researchers/students:** They use the PV monitoring system to test their solar modules. Researcher develop PV modules, so they might want to test them to get the performance of their product. Students may follow courses in which they also create solar modules and thus also need to test their modules. The users will have the ability to test their solar modules on this new field test, which will be capable of testing most types of solar modules. Next to the PV performance data, the setup will provide the available weather data.
- **PVMD Group:** They are currently responsible for the PV test site. They would like to give the opportunity to researchers and students to test all sorts of PV modules while providing a complete picture by providing weather data. After the project is done, they will be maintaining the setup. They will keep the system operational and assist students with setup. So they need good documentation to understand the new system.
- **TU Delft:** They provide subsidies to realise the new system. Additionally, they provide the location for the database and the entire setup.
- **PV community:** Although not influenced directly, they might benefit from better research of PV modules, which are tested on the new setup. Additionally, with proper documentation, the test setup could be replicated at other locations.

2.2. Project Scope and Objectives

This section will analyse the needs of the main stakeholders. From these needs objectives and criteria can be set which can be used to set requirements. The two main stakeholders that are discussed are the students/researchers and the PVMD group.

2.2.1. Students and Researchers

First, the project scope and objectives for the researchers and students are discussed. They will work with the new system. The needs of the students and researchers can be seen in Table 2.1.

Table 2.1: Scoping table analysing needs of the students and researchers

Needs	Objectives	Criteria
Accessibility	Make accessing and processing data easy.	Measurement files must be easily accessible by authenticated users by running a script on their own laptop.
Reliability	Make sure the system is always online.	The system must have an uptime of 99%.
Data	Make sure the system measures and stores all relevant data.	The system must include the voltage (V) and current (A) readings of each measurement with all available weather data and meta-data about the module.

Researchers and students are the end users. The thing for them that is very important is that the system is intuitive and easy to use. The interface should be as intuitive as possible. To make it easier, a manual will be created for additional support. Besides, researchers want to get their test results quickly and reliably, which means that the system needs to work nearly always unless some mandatory maintenance is needed. One of the most important things for the users is the data. Researchers and students alike want to get the varying situations in which their system operates. This means collecting data for these situations. A weather station is already in use at the test site, so all the information this station gives will be added to the database to give full insight for the users.

2.2.2. PVMD group

Next, the needs of the PVMD group are discussed. The needs can be found in Table 2.2.

Table 2.2: Scoping table analysing needs of the PVMD group

Needs	Objectives	Criteria
Reliability	Make the system so that it has low downtime.	Members of the PVMD group must be automatically notified when and what kind of fault occurs.
Documentation	Make the system well documented.	The documentation must be documented in a way that any member of the PVMD group would be able to independently replicate the design.

Since the PVMD group will be the ones who own the product, they need to maintain it. This means that for them to be able to do that, they need good documentation so that they can easily understand the system. They also want the system to need as little attention as possible, and for it to just work. As stated by the previous stakeholder, the system must have a 99% uptime. To reduce potential downtime, members of the PVMD group must be notified when and which fault occurs so that they can fix any issues.

2.3. Functions

Following the needs of the stakeholders, the functions of the system can be derived. The functions are the things that the system is expected to be able to do. These functions are listed below.

- Allow the user to input all the details of the solar module.
- Allow the user to input details on the measurement type and interval.
- Connect to multiple measurement devices on the same bus.
- Schedule a measurement based on the user input.
- Perform a measurement of the PV module at the given time.
- Stop measuring after the given stop time.
- Collect Weather data from the weather database of the existing setup.

- Combine the PV and Weather measurement data.
- Store the PV measurement results into a CSV file.
- Add the information from the CSV file to the database.
- Let users access the database to retrieve their data as a CSV file.
- Notify members of the PVMD group about the status of the system.
- Monitor the temperature of the new power dissipation device.

2.4. List of requirements

Following the functions and discussing with the supervisors, a list of requirements has been set up. Each requirement has been categorised into different categories and is given an ID for later reference. The categories that are defined are interface, usability, reliability, security, functional and performance.

Table 2.3: List of all the requirements sorted by category

Category	ID	Requirement
Interface	INT-1	The new database shall have the following column headers, which come from the user: module_name, mounted_on, axis_azimuth, axis_tilt, user_name, user_email, area, technology, and manufacturer.
Interface	INT-2	The new database shall have the following column headers, which come from the OPET tracer and the supervisor Python file: date_time, scheduled_time, v, i, temperature_cell, and status_integer.
Interface	INT-3	The CSV file shall have the following column headers from the weather database: temperature_air, relative_humidity, dew_point, relative_pressure, wind_speed, wind_speed_std, wind_direction, wind_direction_std, and irradiance.
Interface	INT-4	The measurement JSON input file shall contain module attributes such as module_name, mounted_on, OPET-ID, interval_point, interval_curve, stopdate, disabled, load_mode, user_name, user_email, area, technology, and manufacturer; rack attributes such as axis_azimuth and axis_tilt; global attributes such as data_destination, admins_email and lookback.
Interface	INT-5	The users shall get the output as a CSV file.
Interface	INT-6	The system shall access the SQL weather database over the network to retrieve weather data.
Interface	INT-7	The system shall be installed on the PC inside the monitoring station.
Interface	INT-8	The OPETs shall be daisy-chained and connected to the main computer.
Usability	USA-1	The documentation shall contain an explanation of what every function does.
Usability	USA-2	The documentation shall contain an explanation of the overall operation.
Usability	USA-3	The documentation shall contain a manual guiding the user on how to set up a measurement.
Usability	USA-4	The database shall be documented such that a member of the PVMD group can use and recreate it.
Reliability	REL-1	The system shall have 99% uptime.
Reliability	REL-2	The system shall work for at least 10 years.

Category	ID	Requirement
Reliability	REL-3	The system shall keep measuring and storing PV measurement data even if there is no connection to the weather station.
Security	SEC-1	The system shall only be accessible to authorized users.
Functional	FUN-1	The OPETs shall be connected over RS-485 bus to the supervisor.
Functional	FUN-2	The supervisor shall only do measurements before the specified stop date.
Functional	FUN-3	The supervisor shall do the measurements forever if no stop date is specified.
Functional	FUN-4	The supervisor shall be able to do measurements for short circuit current, open circuit voltage, and maximum power point tracking.
Functional	FUN-5	The system shall be able to perform a point measurement of a module.
Functional	FUN-6	The system shall be able to measure the I-V curve of a module.
Functional	FUN-7	The system shall combine the weather data with the module measurement.
Functional	FUN-8	Users shall be able to retrieve all point or curve measurements from a specific module within a specified date range.
Functional	FUN-9	Once a day, the system shall try to upload the missing measurement data if uploading failed during the day.
Functional	FUN-10	Maximum once a day, the PVMD group shall be emailed in case something is malfunctioning.
Functional	FUN-11	The firmware shall be updated to read the temperature from the MAX31856.
Functional	FUN-12	The firmware shall be updated to communicate the information from the MAX31856 to the supervisor.
Functional	FUN-13	The firmware shall monitor the temperature of the power dissipation device by reading the ADCs.
Functional	FUN-14	The firmware shall disconnect the output when the temperature of the power dissipation device is above a configurable temperature.
Functional	FUN-15	The firmware shall be able to disconnect the output in case not all ADCs respond.
Functional	FUN-16	The firmware shall be able to communicate the temperature readings to the host.
Performance	PER-1	The system shall be able to do at least one point or curve measurement every minute for each module.
Performance	PER-2	The delay between the scheduled measurement and the measurement time shall be less than one second for a point measurement.
Performance	PER-3	The delay between the scheduled measurement and the measurement time shall be less than five seconds for a curve measurement.
Performance	PER-4	Between 20 °C and 75 °C steady state, the temperature measured by the power dissipation device shall not deviate more than 5 °C from a calibrated reference sensor.

Alongside the requirements that resulted from the case study and talking with the supervisor, some requirements resulted from the other subgroup. This subgroup will install a power dissipation device. The temperature of this device shall be monitored. The output shall be disconnected when the temperature is too high or when not all ADCs respond.

3
Materials & Methods

3.1. Setup

Before diving into more detail, the setup is discussed first. A visual representation of the setup is shown in Figure 3.1. Solar modules will be installed on the racks. Cables connecting these solar modules are connected to the OPETs. These OPETs collect the data from the solar modules by performing measurements. The results of these measurements will, in turn, be sent to the database. This is done via locally stored CSV files. There is also a weather station which collects data about wind speed, temperature, irradiance, etc. This weather data gets stored on an older database running on MySQL. The new database collects some of the weather data from the MySQL database over the network. The new database manages these different files and enables users to get outputs. The user can get an exported CSV file that contains the OPET measurements and the weather measurements.

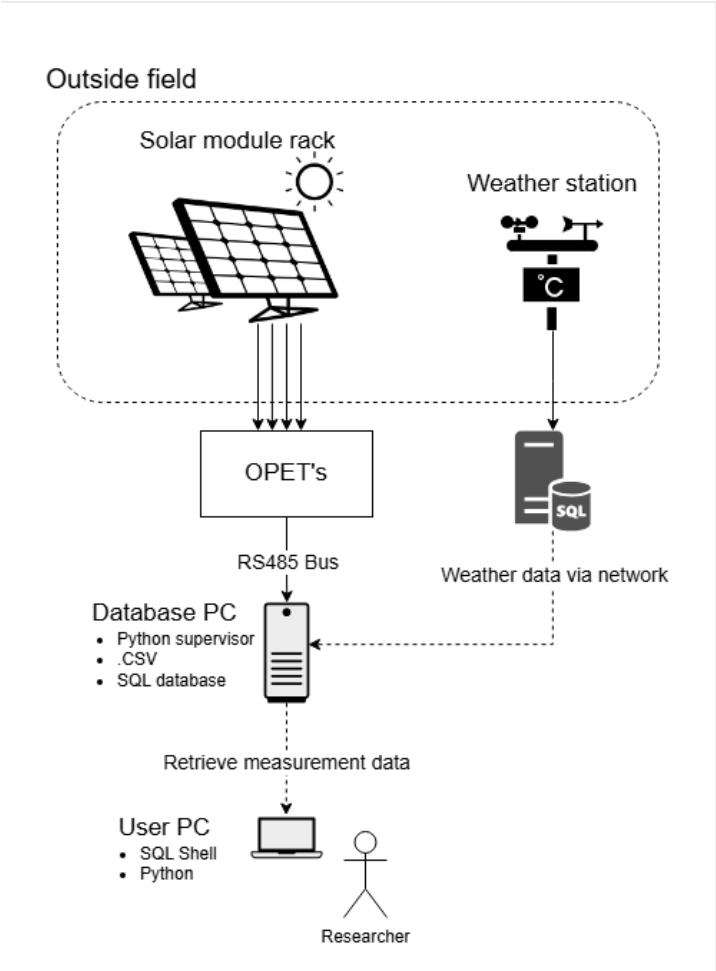

Figure 3.1: Illustration of the new measurement and database setup

The three things that need to be implemented are: the database (Section 3.3) which store the measurement results, the supervisor (Section 3.4) which schedules the measurements and communicates with the OPET, and the firmware (Section 3.5 and Section 3.6) which discuss the new developed firmware which include temperature monitoring.

3.2. Writing software

For this project, software needs to be developed. Before diving into what has been developed, some design choices need to be made on how to write the code. That will be discussed in this section.

3.2.1. Coding style guidelines

This project mostly revolves around setting up software to get the new monitoring station started. After this project, the members of the PVMD group will maintain and update the codebase. To ensure easy maintainability, writing the code should be done following some guidelines. For writing Python code, the PEP-8 coding standard will be followed as it provides a clear, readable coding standard [12].

Additionally, some C code needs to be developed to update the firmware. The provided firmware does not follow a strict coding guideline. Therefore, the newly written code will try to mimic the existing writing style to ensure consistency.

3.2.2. Naming conventions

The database will store parameters related to PV modules. It was suggested to use the standard set by DuraMAT [13]. They provide a guide on how to name PV-related terms or abbreviations. This ensures consistency across multiple research institutes in relation to naming PV-related terms.

3.2.3. Software development

Some parts of the project build upon existing GitHub repositories. As the new firmware will be developed upon the existing implementation. Additionally, some extra features are desired for communication with the OPET. For that reason, the opet-firmware and opet-control repositories are forked [14], [15]. This allows for extending these repositories with functions suited for the new use case, and if desired by the original owner, can be included in the original repository.

In addition, a new GitHub repository is set up to create the supervisor and database communication [16]. It will include documentation of how the database is set up and how to run it. For researchers, there are some Python scripts that will help them receive their data and plot it.

3.2.4. Error handling

During operation, many errors can occur. These can come from connection errors with the OPET, human-made errors in configuration files, or race conditions. To ensure the reliability over a longer period of time as required under `REL-1` and `REL-2`, these errors should be caught using try and except statements. This way, if a part fails, proper follow-up action can be taken and logged. Logging these errors will allow the maintainers to check what is going wrong at what time.

3.2.5. Documentation

To enable others to use the database and understand the software, some documentation had to be created. The documentation includes information on the structure and functions as required under `USA-1` and `USA-2`, a user guide specified under `USA-4`, and a setup guide as required by `USA-3`. The documentation can be found on GitHub TUD-PV-monitoring-tool [16].

3.3. Database

The role of the database is to store the data in an organised manner. As previously said, the export of the OPET supervisor is a CSV file. The name of that CSV is based on the date the measurements were taken and the type of measurement. There are 2 types of measurements: curve and point. This data structure is the foundation of how the system works. It also collects the data about the weather from the MySQL database. This data is collected over the network. These 2 different types of measurements have to be added to create a coherent database.

In this section, various design problems will be discussed. First, how the network is set up and how it should be set up. Then what type of database software was used, and how the database is structured. Following that, the inner workings of the database are explained. First, the inner core and then some other important functions. Lastly, some data extraction tools are explained.

3.3.1. Networking

The connection to the old weather database has to be established via port forwarding and network access. The weather database is not on the standard TU Delft network, but is located on a specific network to isolate the machine from the TU Delft network. This means that it is not easily accessible via port forwarding due to the firewall of TU Delft. When the PV system is officially installed, the new database should be connected to 2 networks. It has to be on the official TU Delft network so that it is easily accessible via the IP address. It also has to be on the same network as the weather database, so that the new system can easily access it. However, because the firewall cannot be easily changed, there needed to be a solution that is meant for testing purposes. For this, Tailscale is used to create a VPN network including various devices [17]. With this, a subnet needs to be created on the new database and on the weather database. This allows anyone with access to the VPN network to access the IP addresses of these 2 devices. So by installing this, the firewall can be bypassed, and the database is still accessible from user machines.

3.3.2. PostgreSQL

The database is built on PostgreSQL [18]. The reasons why PostgreSQL was chosen are significant. The first advantage of PostgreSQL is that it is relational, so that tables can easily be linked together [19]. This gives the ability to keep tables relatively comprehensible and still save a lot of data. The most important reason for PostgreSQL is that it is widely documented; the same goes for MySQL; however, Oracle Database is much less documented on the internet. This makes it easier to learn, which, in the timespan of 10 weeks, is a major advantage. PostgreSQL is highly extensible, has great scalability, and has strong data integrity compared to MySQL [20].

3.3.3. Structure

The database consists of 4 tables as seen in Figure 3.2. The data that is collected from the OPETs is sent to the `pv_point` and `pv_curve` tables. The `pv_point` table consists of measurement time, scheduled measurement time, module name, on which rack it's mounted, voltage, current, cell temperature, status integer, azimuth angle, tilt angle, and the weather id as specified by `INT-2`. The `pv_curve` table saves current and voltage as vectors and adds measurement duration to this, because it takes 100 measurements. Measurement duration has to be used, because 100 measurements cannot be done at a specific time. There is a limitation on the information that can be uploaded to these tables: the combination of `date_time` and `module_name` must be unique. A single solar module can only produce a single measurement at a specific time.
The data that is collected from the weather station database goes into the `weather` table. The primary key `weather_id` is an integer, because the original serial from the weather database is used. This makes a user able to check if the data in the database is consistent with the weather database and possibly find even more data about the given measurement. The `weather_id` links the OPET measurements to the weather measurements as specified by `FUN-7`.
The last table is the `modules` table. In this table, information about the PV module is saved. It contains information about which OPET tracer the module is connected to, so that the OPET is identifiable. It also contains information about the user and their email address, because if there are problems with their PV modules, they can be found. Also, some information about the technology and the manufacturer is stored. All of this is in line with `INT-1`. The `pv_point` and `pv_curve` get linked to the modules

table via the `module_name` attribute.

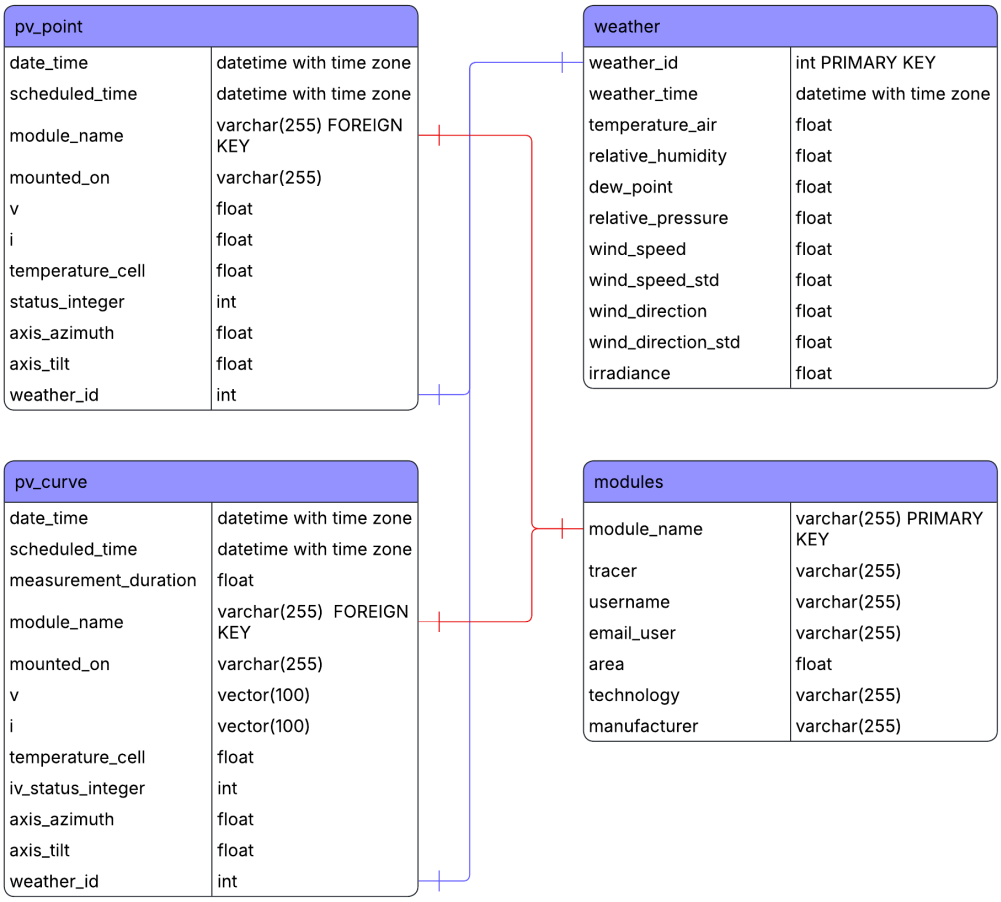

Figure 3.2: Structure of the PostgreSQL database with datatypes

3.3.4. Operation of the database

The database has multiple functions that upload the collected data to it. This is done via the function `update_loop()`, which can be found in Section D.1.1 and `daily_loop()`, which can be found in Section D.1.2. The difference between these two is that one is constantly running, and the other only runs once a day. Both of these functions are started by the supervisor.

First, the `update_loop()` uploads newly collected data every 10 seconds. Every 10 seconds is more than frequent enough to keep a database up to date. Because realistically, point measurement probably will not be made at a higher frequency than one measurement every 30 seconds. The way that it works is that it inserts the OPET data that was collected on the day that it is via the `add_data()` function. So, for example, if the date is the 30th of May in 2026, it will upload the CSV files with that date in the name. It also constantly checks if new modules have been added to the `measurement_config.json`. These modules will then be uploaded to the `modules` table. For the weather data, it first pulls the data from the weather database that was made in the past 24 hours. This then gets inserted into the `weather` table. If the program is unable to upload the data from the weather database, it will log the error, and importantly does not stop the other operations from happening. This means that OPET data can still be uploaded even when the weather data cannot be inserted, as specified by `REL-3`. The same goes for when the system is not even able to connect to the weather database. When all the data has been collected, it will later try to give a `weather_id` to the OPET measurements. An explanation on how this `weather_id` is assigned will be provided in Section 3.3.6

The daily_loop() has a different function compared to the update_loop; it uploads data retroactively. It does this once every 24 hours. If the database has been offline for a couple of days and the OPET measurement data has still been collected, it can retroactively be added to the database as specified under FUN-9. It does not do this until the end of time, because that would make the operation take increasingly longer as time goes on. That is why there is a setting in the measurement_config.json that is called lookback, which sets the number of days to which the system looks back and uploads retroactively. It uses the same techniques as the update_loop for the OPET measurements and the weather_id assignment. However, the weather data is collected in a different way. It is done using a single query. It asks for all the data that is in the weather database since the lookback number of days ago.

3.3.5. Error Detection

The daily_loop() also contains another function that differs from adding data; it has an error detection system, which can be found in Section D.1.5. The system checks every 24 hours for problems, and if one is found, it sends an email to the admins as specified by FUN-10. The first way it does this is by checking whether the system has received any new data for both the point and curve measurements in the past 24 hours. If this is not the case, it will send an email to the admin that the entire system has not received new information in the past 24 hours, and the datetimes of the last measurements for the curve and point. If it has received data for one of the measurement types, it will send a warning that it has not received any data for the other measurement type, and when that last measurement was received.
The other error detection systems check the performance of each module. The first way it does this is by reading the number of errors in the past 24 hours for every module. The OPET measurement contains a status bit. If it is 1, nothing is going wrong; if it is anything else, there is a problem. The system counts the number of times this error bit was not 1 and sends the number of measurement errors relative to the number of measurements to the admins of the system and the PV module user.
The last form of error detection is that the database checks whether new data is being collected for every PV module. It works in the same way as the error detection that checks whether any data is being uploaded, but this time, a filter is applied over the module name. It also considers whether the PV module is active, i.e., disabled=False.
The emails are being sent via SMTP, which is a common and easy way to make computers send no-reply emails to users. The emails are encrypted via TLS, so people cannot read the message over the network.

Outside of the function that detects errors, there is another way that errors can be detected. Because nearly every error gets logged into the logfile, so if there is a problem with a specific function, information can be found on what went wrong in the logfile.

3.3.6. Weather ID

To connect 2 different measurements, a system that assigns a weather_id to the OPET measurements had to be made, which can be found in Section D.1.4. The OPET measurements and the weather measurements are not done synchronously, which makes it quite tricky to combine these 2 measurement and still keep the data representative. The way the module_id is assigned is done with a slight delay of 5 minutes. This is done because the system wants to know what the closest measurement is within a 5-minute region of the OPET measurement. This region is only 5 minutes, because then the weather data is still representable for the timestamp. If the time gap were to deviate any further, the weather may have changed significantly compared to when the OPET measurement was made. If no weather measurement is within the 5-minute region weather_id 0 will be assigned.
An alternative to this solution would be to assign the most recent weather_id, which would be easier to implement. This would give an immediate weather_id when the OPET measurements were uploaded to the database, but the corresponding weather measurements may have been a lot less representative. So the delay of 5 minutes seemed worth it.
Another solution could have been to interpolate all of the weather measurements so that the OPET measurement would be aligned with an estimated weather measurement. This, however, had one significant downside. The weather_id that is used is generated by the weather database. This means that if the measurements are interpolated, the connection to the weather database is lost, because the

weather ID is not interpolatable. So the weather measurement cannot be found in the weather database, because the `weather_id` that exists for the OPET measurement does not exist in the weather database.

3.3.7. Module Data

The module data gets updated via the `update_loop`, so basically the system reloads the configuration file every 10 seconds and checks if there are any changes. The code can be found at Code Listing D.5. The module data, once it has been added, is not changeable. This is because once a module is made under a specific name, it cannot change specifications. For example, a PV module cannot change in size; if it is different in size, it is a different module. However, some things can be changed about the module data as long as the rest of the data is correct for the thing stored in the database. The things that are changeable are tracer (which OPET the module is connected to), username, and user email. This is done because the module can get connected to a different tracer, and thus that needs to be changeable. The username and user email can be changed so that the ownership of a PV module can be changed. For example, if someone leaves the organisation, they might hand some of their projects over to someone else.

3.3.8. Data extraction

One of the most important features of the system is the data extraction tool `download.py`. According to requirement `INT-5`, the user shall receive a CSV output file. The output is a point or curve measurement, joined with the weather data, which is specified under `FUN-7`. The user must select whether it wants to receive the point or curve measurements. To make this action quicker, the system filters for dates in the query, which is required by `FUN-8`. Because it queries only data between the start and stop dates, it does not need to load the entire database. After that, it filters for either a single or multiple PV modules.

Another data extraction tool is `livedata.py`. This program pulls the most recent data from the database for a specified PV module. It continuously plots the current and the voltage. This tool is useful when a new solar module is installed because it shows whether the database is actually collecting data. This program can also be used to check whether the system is online and working. Because if this system does not respond, then something is wrong with the system. Or if the data has been measured a long time ago. Alternatively, the user can use SQL shell to select any data they want.

3.4. Supervisor

The main role of the supervisor is to start each process, schedule measurement jobs, and log any errors. To get started, the team was already provided with a supervisor script that was able to schedule measurements and write the results into a CSV file. This section will explain how the supervisor works, how the jobs are scheduled, and how the measurement loop and writer loop work.

3.4.1. Multiprocessing

The supervisor uses the multiprocessing Python package that allows starting multiple processes at the same time. It must be able to start the two update loops from Section 3.3, namely `update_loop` and `daily_loop`. To do and store the measurements, `measurement_loop` and `writer_loop` are used. There is one writer loop process started, and for each RS-485 bus, a measurement loop process started. Each process also gets a name assigned corresponding to its process. This is used to check in the main loop if any process has died. If this is the case, a new process can be started with the correct function. Full implementation of the supervisor can be seen in Section D.2.1.

The supervisor also sets up a shutdown event. Each process will check the status of the shutdown event. If it is set, it will complete the last iteration to exit cleanly. So instead of setting up infinite while loops as:

```
1 while True:
```

Processes can use:

```
1 while not shutdown_event.is_set():
```

Using this lets processes finish their tasks when an interrupt or error occurs. For example, the writer loop can still finish writing the last results.

3.4.2. Jobs

Every 30 seconds, the supervisor will schedule new jobs for the measurement loop to do measurements. Each module configured in the measurement configuration file is checked if it should be disabled or if it has passed the stop time. The stop time is a date that specifies after which no more measurements need to be taken. Users can also configure this to `null`, which schedules that module indefinitely. If the module is inactive, a load job is scheduled to disable the output of the OPET.

The supervisor will schedule 3 different jobs for the measurement loop: `set_load_mode`, `point`, and `curve` as required by `FUN-4`, `FUN-5`, and `FUN-6`. The shared dictionary `jobs` will contain all the information for the measurement loop to be able to do the measurement, such as the scheduled time, OPET bus, and address, and the job type.

Only modules that are not inactive are scheduled to do point or curve measurements. The supervisor looks at `point_interval` and `curve_interval` from the measurement configuration to determine the scheduled times for the measurements.

When all jobs are scheduled, the supervisor checks that all the scheduled jobs are not too old. The supervisor might lose connection with one of the OPETs. In that case, the supervisor keeps scheduling jobs, which could result in a job queue that grows large over a longer period of time. Therefore, if the scheduled time of a job is more than 5 minutes in the past, it is removed.

3.4.3. Measurement loop

The measurement loop will first try to initialize the RS-485 bus and start a connection with each OPET connected to that bus as required by `FUN-1`. It will look at the JSON files, which OPETs should be available at the given bus. It will try to make a connection if the OPET connection fails at startup. Once each connection is made, the infinite measurement loop starts. It will look at the shared dictionary with the pending jobs and check if there are any jobs related to the OPETs connected to the bus.

Before the OPET is instructed to do a job, the supervisor first checks if the OPET is available using `.available` from the OPET library. It will then retry the next loop iteration to do the job. The errors will be logged if something goes wrong during communication.

The measurement loop will execute the three different job types: `set_load_mode`, `point`, and `curve` as discussed in Section 3.4.2. `set_load_mode` is used to set the OPET into the correct measurement mode or to disable the output. It is able to set the OPET into short circuit current, open circuit voltage, and maximum power point tracking mode as required by `FUN-4`. After the mode has been set correctly, it will enable the output. Alternatively, if an OPET is inactive, the output will be disabled.

For a point measurement, the measurement is executed using `.sample` from the OPET library. The relevant information of the measurement is stored in the shared dictionary `results`. This relevant information is the headers used for the database and are set in `INT-2`, together with the name of the module and information about the mounted rack.

For a curve measurement, the supervisor first needs to request a curve measurement by using `.start_iv_curve()`. Next, it will check if the measurement is complete by checking if the OPET is available and if the operation is complete by using `.operation_complete()`. If this returns true, the IV data will be read using `.iv_data`. Otherwise, the OPET will be checked in the next loop iteration. The result will again be stored in the shared dictionary, similar to the point measurement. The curve measurement only has a different status integer.

There are many parts in the measurement loop that could give errors. For example, the OPET could give a timeout error, or there is a race condition where a job is already removed by a different process. Since the supervisor should be reliable for a long period of time as required by `REL-1`, `REL-2`, the program should catch these errors, log them, and continue. It is important that other measurements continue even if one fails.

The entire main loop is set behind a try statement. If it still crashes for a different reason, it will raise an exception. This will allow the main supervisor to restart the process. Additionally, the serial port is closed to avoid it being left open. Full implementation of the measurement loop can be seen in Section D.2.2.

3.4.4. Writer loop

To store all the measurement data collected by the measurement loop, a writer loop is used. This loop will pop one by one a result from the shared dictionary `results` and write it into a CSV file. A new CSV file is generated for each day. If the shutdown event is set, the loop will first finish writing into the CSV file. The implementation of this writer loop can be seen in Section D.2.3.

3.4.5. OPET control library

The OPET control library provides the necessary functions that are used to communicate with the OPETs. Most can be used directly. However, it does not retrieve the temperature of the solar module if it is available. Therefore, a small modification is needed to include it. This modification for the sample function can be seen in the Code Listing 3.1.

Code Listing 3.1: Included module temperature with a point measurement

```
    @property
    def sample(self):
        '''Dictionary with a single measurement sample'''
        keys = [
            'status_integer',
            'voltage',
            'current',
            'voltage_offset_internal',
            'voltage_bias',
            'temperature_1',
            'temperature_2'
        ]
        measurement_time = datetime.now().astimezone()
        reply = self.send_verify('READ?')
        if len(reply) == 8: #If longer reply, module temp is available
            keys.append('temperature_cell')
            reply = dict(zip(keys, reply))
            for key in keys[1:]:
                reply[key] = float(reply[key])

```

```
        else: #Temperature cell not available, set to none
            reply = dict(zip(keys, reply))
            for key in keys[1:]:
                reply[key] = float(reply[key])
            reply['temperature_cell'] = None

        reply['power'] = reply['voltage']*reply['current']
        reply['status_integer'] = int(reply['status_integer'])
        reply['measurement_time'] = measurement_time
        reply.update(self.parse_system_status_integer(reply['status_integer']))
        reply.update({'address_integer': self._address_integer})
        return reply
```

It looks at the length of the reply to determine if `temperature_cell` is available. The same approach is used to include the temperature with the IV curve data, which can be seen in Code Listing 3.2.

Code Listing 3.2: Included module temperature with the IV curve data

```
    @property
    def iv_data(self):
        '''Stored I-V curve data from the latest IV or transient measurement.
        The measurement is updated using `start_iv_curve()`.'''
        result = {}
        reply = self.send_verify('IV:DATA?')
        result['iv_status'] = int(reply[0])
        if len(reply) % 2 ==1: #Uneven amount of results, no temp data
            result['temperature_cell'] = None
            result['voltage'] = [float(value) for value in reply[1::2]]
            result['current'] = [float(value) for value in reply[2::2]]

        else: #even amount of result, temp data available
            result['temperature_cell'] = float(reply[1])
            result['voltage'] = [float(value) for value in reply[2::2]]
            result['current'] = [float(value) for value in reply[3::2]]
        return result
```

3.5. Module temperature monitoring

To give more insight into the performance of the solar module, the temperature of the module is also measured. To measure this, the PVMD group already had some T-type thermocouple sensors available. To read out these thermocouple sensors, the MAX31856 is used as specified in `FUN-11` [21]. This sensor will be able to measure the temperature of a T-type thermocouple and communicate the result over SPI. The user can already configure a temperature sensor in the existing firmware. The OPET uses the Atmel ATmega1284 8-bit microcontroller, which uses the AVR enhanced RISC architecture and has on-chip flash memory [22]. In the `main.h` file, the user can configure characteristics about the setup. It uses a define statement for `PCBconfig_TEMP_is_enabled` and `PCBconfig_TEMP_Sensor_Type` to enable and configure the temperature sensor.

However, the MAX31856 is not one of the supported sensors. This new sensor also needs to be defined in the main header file. The other sensors have two functions in order to give temperature measurements: one for setting up the sensor and one for retrieving measurements from the sensor. For the MAX31856, these functions need to be made.

The MAX31856 uses SPI to communicate the readings to a microcontroller. SPI, or the Serial Peripheral Interface, is commonly used for synchronous, full-duplex communication between a master and peripheral units. The master initiates communication by pulling the slave select pin low and providing a shared clock signal. It uses two data lines, MOSI and MISO, to achieve full-duplex communication. Communication is finished by pulling the slave select line high [22].

3.5.1. Setup

The MAX31856 has two configuration registers that need to be loaded with the correct values. The first configuration register is at address 0x80. In the setup, it is configured to set the sensor in automatic conversion mode and enable the 50Hz noise filtering. The second configuration register is located at 0x81. Here, the type of sensor and the number of samples used to average can be configured. As the

temperature is not monitored often (maximum every 30 seconds), the maximum number of averaging is used, which is 16 samples. Additionally, the sensor is set to a T-type sensor [21].

There already is a function that uses SPI to set up a different temperature sensor. To implement it for the new sensor, the same approach is used. Full implementation of the setup can be seen in Section C.1.

3.5.2. Measure

With the sensor set up, the microcontroller can request temperature readings. The MAX31856 has a 19-bit resolution, including a sign bit. For reading the temperature, three bytes should be read. To achieve that, the chip select is pulled low, and the address 0x0C is sent. This is the address of the low byte. After sending the address, a dummy byte is sent to receive the low byte. Afterwards, two more dummy bytes can be sent to receive the other two bytes, which can be read successively without sending the other addresses. Converting the bits into a temperature can be done by Code Snippet 3.3. It removes the lowest 5 bits as the temperature is stored in the other 19 bits. Secondly, if the temperature is signed, the other higher bits should also be set to one to preserve the sign. Full implementation can be seen in Section C.2.

Code Listing 3.3: Converting a raw reading from the MAX31856 into a temperature

```
raw = ((int32_t)LTCBH << 16 ) | ((int32_t)LTCBM << 8 ) | LTCBL;
raw >>=5; //Remove 5 lowest bits, temperature is only 19 bits including 1 sign bit

//If signed
if (raw & 0x40000) {
        raw |= 0xFFF80000;
}
float Temperature = raw * 0.0078125;
```

To communicate the information from the MAX31856 as stated by `FUN-12`, these functions are included in `main.c`. It uses an infinite while loop with different timers to make measurements or handle communication. There already is a timer to measure the temperature, which triggers around every 5ms. The result is stored in `AI_RTD_Temp`. This variable stores the latest module temperature and sends it over RS-485 when the `READ?` command is sent. The communication of the curve data is also adjusted to give the temperature of the module.

3.6. Power dissipation device

The firmware should also be configurable such that it can monitor the temperature of the new power dissipation device. Before diving into the details of the firmware, background information about the dissipation device is discussed.

To ensure the high current OPET will be able to dissipate up to 800W, the other subgroup designed a device that will be able to dissipate it. It uses a modular design with MOSFETs. Up to 8 of these devices can be connected together. Each separate device has a MOSFET, of which the temperature should be monitored. To monitor it, the hardware subgroup chose to use a voltage divider with a fixed resistor and an NTC thermistor as shown in Figure 3.3. This NTC will be mounted on the MOSFET.

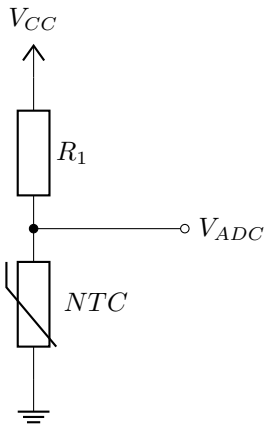

Figure 3.3: Circuit diagram of the temperature monitoring device using the ADC and NTC

The resistance of the NTC can be calculated by reading the voltage with an analog-to-digital converter (ADC). The voltage of the ADC can be written as in Equation (3.1).

$$V_{ADC} = V_{CC} \cdot \frac{R_{NTC}}{R_1 + R_{NTC}} \tag{3.1}$$

Equation (3.1) can be rewritten to obtain the resistance of the NTC as shown in Equation (3.2).

$$R_{NTC} = \frac{R_1 \cdot \frac{V_{ADC}}{V_{CC}}}{1 - \frac{V_{ADC}}{V_{CC}}} \tag{3.2}$$

The resistance value of the NTC is related to its temperature. This is described in Equation (3.3) [23].

$$R_{NTC} = R_{T_0} \cdot e^{B(\frac{1}{T} - \frac{1}{T_0})} \tag{3.3}$$

Where R_{T_0} is the resistance of the NTC at the reference temperature in Kelvin, B is the beta constant in Kelvin, T is the temperature in Kelvin, and T_0 is the reference temperature in Kelvin. Equation (3.3) can be rewritten into Equation (3.4) to calculate the temperature T as function of NTC resistance R_{NTC}.

$$\frac{1}{T} = \frac{1}{T_0} + \frac{\ln(\frac{R_{NTC}}{R_{T0}})}{B} \tag{3.4}$$

The hardware team has chosen the NTCALUG01A103J NTC thermistor from Vishay, which has a B value of 3984 K, a reference resistance R_{T0} of 10 kΩ at a reference temperature of 298.15 K, and a fixed resistance R_1 of 10 kΩ [24]. Using these equations, two helper functions are created to convert from voltage to temperature and vice versa.

The new functions created for monitoring the temperature can be found in Appendix C. For reviewing the full firmware implementation, see the GitHub repository [14].

3.6.1. Setup

The hardware team had chosen to use the ADC121C021 analog-to-digital converter from Texas Instruments to read the voltage. This is a 12-bit ADC equipped with an alert pin, compatible with the I2C protocol, and has 9 hardware-configurable addresses [25].

The I2C protocol can connect up to 128 devices and uses a shared clock (SCL) and data (SDA) bus. Each device connected to the bus has an individual address to which the device will respond. One address packet is 9 bits long: 7 bits correspond to the device address, 1 bit encodes read or write interaction, and the last bit is an acknowledge bit. The first 8 bits come from the master, and the device acknowledges by pulling the SDA line low [22].

The number of ADCs may vary, as each configuration might use a different number of MOSFETs. The user must make sure that it does not use the same address twice in one configuration. At startup, the microcontroller will check whether it receives a response on all the possible configurable addresses. It does this by sending each possible address and checking if it receives an acknowledgement. If the ADC is present, a setup function is called that will configure the ADC. If the ADC is present, `AI_HEAT_ADC_Present_Mask` mask is updated. This mask is used later to read from ADCs that are active. Also, the fault mask and the temperature are initialised, which can be seen in Code Listing 3.4.

Code Listing 3.4: Setting up the ADCs for monitoring the temperature

```
void Temp_monitoring_Setup() {

    AI_HEAT_ADC_Present_Mask = 0;
    AI_HEAT_ADC_Fault_Mask = 0;

    for (uint8_t i = 0; i < 8; i++) {
        if (ADC121C021_Is_Present(ADDRESSES_ADC121C021[i])) {
            AI_HEAT_ADC_Present_Mask |= (1 << i);
            ADC121C021_Setup(ADDRESSES_ADC121C021[i]);
        }

                AI_HEAT_NTC_Temp[i] = -9.9;
    }
}
```

The `ADC121C021_Setup` calls three other functions to set up the configuration, low voltage alert, and alert hysteresis registers. The configuration register is set up to enable the alert and set it to auto-clear. Shutting off the output can be based purely on the ADC readings and converting them to temperatures. However, using the alert allows for extra redundancy. Automatic conversion mode is set to the slowest possible configuration of 0.4 ksps, as the ADC is not accessed that often.

Because the setup uses an NTC, the ADC gets configured with a low limit. If the ADC drops below this limit, the temperature is too high. A new EEROM variable has been created for the user to configure this upper temperature limit, which is by default 70 °C. This allows the user to change this limit by writing a different value in EEROM. Using a helper function, the temperature can be converted into the low voltage limit and sent to the ADC.

When the alert is set, the voltage is below the low threshold. Afterwards, it should move above this low limit plus the value set in the hysteresis register. For this implementation, the temperature should drop by 5 degrees in order to release the alert. Again with the helper function, this temperature difference is converted into a voltage, which is set into the hysteresis register. How this is done can be seen in Code Listing 3.5.

Code Listing 3.5: Calculating the hysteresis voltage based on 5 degrees below the temperature limit

```
float temp_hyst; // Temperature where the alert should be turned off
uint8_t msb; // Most significant byte
uint8_t lsb; // Least significant byte
uint16_t Vhyst;

temp_hyst = TEMP_MAX_Heat_NTC - 5.0; // Take 5 degrees below max temperature
Vhyst = NTC_TempToRaw(temp_hyst) - NTC_TempToRaw(TEMP_MAX_Heat_NTC);
Vhyst &= 0x0FFF;
lsb = (uint8_t) (Vhyst & 0xFF);
msb = (uint8_t) (Vhyst >> 8);

// Start I2C communication
// ...
```

3.6.2. Temperature monitoring

As set by `FUN-13`, the microcontroller should monitor the temperature of the MOSFETs. After properly configuring the ADCs, they can be included in the main measurement loop. The main loop triggers an interrupt every 5 ms. Once this interrupt is triggered a hundred times (after 500 ms), the temperature interrupt flag is set, and a temperature measurement is performed. To avoid waiting for 9 devices (8 ADCs and one temperature sensor for the module temperature) to send over the data, only one device is measured at each temperature interrupt. Measuring all nine potential devices can lead to a main loop timer overrun when the measurement takes over 5 ms. It alternates between measuring the module temperature and one of the ADCs. So the module temperature is updated every 1 second, and one of the eight MOSFETs' temperatures is updated every 1 second. Therefore, it will take 8 seconds before all temperatures are updated.

After one temperature is measured, it checks all the temperatures to see whether it is too high. In the case when an over-temperature event is already happening, it checks a lower value to avoid rapid on/off behaviour. If the temperature is still too high, the `HEAT_NTC_Over_Temp` error bit is set. If the alert pin is active, the `HEAT_NTC_Alert` error bit is set. While the temperature of a specific device is only updated every 8 seconds, the alert pin is checked every second.

Additionally, if the temperature is unreasonably low (-10 °C or lower), the ADC is considered as faulty, as this is not expected. This process can be seen in Code Listing 3.6.

Code Listing 3.6: Main loop implementation of the temperature monitoring

```
if ((AI_HEAT_ADC_Present_Mask & (1 << TEMP_meas_i))) {
        Read_Temp_ADC121C021(TEMP_meas_i);
}

// Update ith device for next iteration
TEMP_meas_i++;
if(TEMP_meas_i >= 8){
        TEMP_meas_i = 0;
}

// Set temp flag if temperature is too high
uint8_t temp_flag =0;

for (uint8_t i = 0; i < 8; i++) {
        if (!(AI_HEAT_ADC_Present_Mask & (1 << i))) {
                continue;
        }
        // If over temp is active check a lower limit to avoid rapid on/off
        if(is_Status_HEAT_NTC_Over_Temp){
                if (AI_HEAT_NTC_Temp[i] >= (TEMP_MAX_Heat_NTC -5.0) ) {
                        temp_flag = 1;
                }
        }
        else{
                if (AI_HEAT_NTC_Temp[i] >= TEMP_MAX_Heat_NTC) {
                        temp_flag = 1;
                }
        }
```

```
        // If temperature is below -10.0 degrees set offline as probably something is wrong
        if (AI_HEAT_NTC_Temp[i] <= -10.0) {
                AI_HEAT_ADC_Fault_Mask |= (1 << i);
        }
}

if(temp_flag){
        SET__Status_HEAT_NTC_Over_Temp;
}
else{
        CLR__Status_HEAT_NTC_Over_Temp;
}

// Check alert pin
if (is_ADC121C021_ALERT_Active){
        SET__Status_HEAT_NTC_Alert;
}
else{
        CLR__Status_HEAT_NTC_Alert;
}
```

The main header files define two status bytes that store the status of various conditions of the device. This includes whether the output is on, if there is an input voltage or current error, if the two NTCs of the OPET are too hot, and more. There is still room to add three new errors related to the power dissipation device. Two errors for an over-temperature event with the ADC reading or the alert pin, and one in the case of a communication error with one of the ADCs.

For safety reasons, if an ADC malfunctions, the output should also be turned off as required under `FUN-15`. Turning the output off, or disabling it means that the solar module will be set to open circuit. Based on the `AI_HEAT_ADC_Present_Mask`, the number of active devices is determined. If this number is lower than the configured number of devices, an error is set. This mask is only set on start-up. If the user wants to resolve this error it should configure the correct amount of ADCs or switch out a malfunctioning one and reset the OPET. If the ADC is correctly configured, but communication fails during a measurement, the corresponding bit in the `AI_HEAT_ADC_Fault_Mask` is set. With these two cases, the `HEAT_NTC_offline` error bit can be set as in Code Listing 3.7. The number of active ADCs can be configured by the user in EEROM.

Code Listing 3.7: Setting the device offline error bit based on number of active devices

```
uint8_t number_active_devices = 0;

// Count number of active devices
for (uint8_t i =0; i<8; i++){
        if (AI_HEAT_ADC_Present_Mask & (1<<i)){
        number_active_devices += 1;
        }
}

// If less devices then expected or ADC fault, give error
if(number_active_devices < TEMP_Heat_dissipation_devices || AI_HEAT_ADC_Fault_Mask != 0 ){
        SET__Status_HEAT_NTC_offline;
}
else{
        CLR__Status_HEAT_NTC_offline;
}
```

When either or both of these errors are present, the output of the OPET should be disabled. The OPET already does this if the onboard NTCs are too hot or if there is a bias voltage error. These two new errors can be added to the statement that checks the status of these errors. This way, the output is turned off in case of an over-temperature event or when communication fails.

3.6.3. Temperature communication

To properly monitor the temperature, the host should be able to read these measurements as set by `FUN-16`. These readings can be retrieved when the host sends a `TEMP?` command. The OPET will respond with the temperature of the two NTCs of the OPET, and if configured, the temperature of the

module and the 8 temperatures of the ADC, followed by the present and fault mask. With this command, the user can retrieve all temperature information that the OPET has. Implementing this mostly revolves around using the existing `READ?` command and adjusting the variables used, which is shown in Code Listing 3.8.

Code Listing 3.8: New communication command which is able to give the temperature of the MOSFETs

```
COM_Copy_To_OutSTR_From_Start("TEMP?");
if (COM_Compare_to_OutStr(Address)) {
        FloatToString(Value, AI_NTC_Temp_1);
        COM_Add_To_OutSTR_with_Sep(Value);
        FloatToString(Value, AI_NTC_Temp_2);
        COM_Add_To_OutSTR_with_Sep(Value);

    if(is_SysConfig_TEMP_On) {
                FloatToString(Value, AI_RTD_Temp);
                COM_Add_To_OutSTR_with_Sep(Value);
        }

        #ifdef PCBconfig_TEMP_Heat_Monitoring_enabled
                for (uint8_t i =0; i<8; i++){
                        FloatToString(Value, AI_HEAT_NTC_Temp[i]); //Send each temperature
                        COM_Add_To_OutSTR_with_Sep(Value);
                }
                itoa(AI_HEAT_ADC_Present_Mask, Value, 10); //Send active mask
                COM_Add_To_OutSTR_with_Sep(Value);
                itoa(AI_HEAT_ADC_Fault_Mask, Value, 10); //Send fault mask
                COM_Add_To_OutSTR_with_Sep(Value);
        #endif

    UART_WriteString (&OutSTR[0]);
        CLR__Status_MainTimerOverRun; // clear timer overrun flag
        goto UART_Execute_Command_END;
}
```

4 Results

To verify the functionality of the developed software, a test plan has been created. To test the software, a bottom-to-top build-up approach was used. The individual functions were tested, and then they were applied to the entire program. The tests are based on the set requirements and test various special cases, such as testing what happens if the user makes a mistake in the configuration. Tests are split into three categories: Database (Section 4.1), Supervisor (Section 4.2), and Firmware (Section 4.3). The tests are presented in tables showing what is tested and whether it has passed or failed. Appendix B goes into more detail, specifying when a test is completed and how the test is conducted. In Section 4.4, the concept demonstrator is discussed.

4.1. Database

The database test results in Table 4.1 indicate that the entire system is working properly. Some additional information about each test was left out and can be seen in Table B.1.

Table 4.1: Tests to verify requirements and functionality for the database.

Test	**Pass/Fail**
An authenticated user is able to retrieve measurement data	Pass
User is able to connect remotely to the database	Pass
The database is able to connect to the weather database	Pass
Curve and measurement data are being added	Pass
Live OPET measurement collected via update loop	Pass
Live weather data collected via update loop	Fail
Adding the past OPET data with a lookback time retroactively	Pass
Adding the past weather data with a lookback time retroactively	Pass
Adding the module data to the database	Pass
Add module data with one element missing	Pass, except for the 'module_name'
Adding the weather data to the database	Partially, this data is from 2025 and before
Sending an email via SMTP Python	Partially, currently only via Gmail server
Detect an error when no data is being received by the database	Pass
Detect the number of errors for a day of measurements per day	Pass
Detect an error when data is not being received from a single module	Pass
Retroactively sync the weather ID of the weather data to the measurements	Pass
Module data can be updated only for tracer, username, user_email	Pass
Module data created with a duplicate name will not get added, and the user gets notified via the log	Pass
If the email address is invalid, it gets logged	Pass
Other measurements still get added to the database when weather data is not connected	Pass
Checking if the database has collected data by receiving the most recent measurement	Pass

All of the tests in Table 4.1 are done on a PC and not on the server. This is because some functionality related to the networking configuration was not completed on time. The results overall are quite positive. The database is only accessible to users with a remote account and otherwise, they cannot get access to the data in the database as required under SEC-1. The database is able to add data retroactively FUN-9. The users are able to download the data as required under FUN-8. However, the results contain some small caveats. The emailing server, as required under FUN-10, currently only works with a Gmail server address. This is due to the fact that no email has been provided yet by TU Delft. However, an SMTP server should easily work for a Microsoft service, so this should not be a significant problem. Another thing that is untestable currently is a test of the complete system with the weather database. The weather database is currently not collecting any new data because of a problem with a bus connector. This means that it is possible to add data from 2025 and before for testing. It is not possible, however, to test whether the entire system works with a live data stream from the weather station. So if the system wants to upload the data for today, including the weather data, that is not possible because the weather data does not exist. The testing that has been done with weather data was either with old data or with synthetically generated data points. Nonetheless, it is still able to upload the OPET data for that date without the weather data.

An extra test has been done to test the performance of the system. In this test, a CSV file has been created that contains 1048576 rows of dummy data. Then, an attempt was made to add this data to the database to measure how long it takes to perform this exercise. This is done for the point measurements because the point measurement CSV file has a lot of entries, while the curve measurement entries are relatively insignificant in size. This file is unreasonably large to test the maximum performance of the system. In reality, if 32 OPETs were connected to the machine, each doing a point measurement every 30 seconds, then a CSV file with 92160 rows would be created. This test took 117.95 and 240 seconds on a PC and on the server, respectively.

4.2. Supervisor

Tests related to the supervisor can be seen in Table 4.2. Additional information about each test can be seen in Section B.2.

Table 4.2: Tests to verify requirements and functionality for the supervisor

Test	**Pass/Fail**
Schedule, do, and save measurements	Partially for two modules.
The system is able to do at least one point measurement every 30 seconds and one curve measurement every 5 minutes for each module within the performance measures set in PER-2, PER-3	Partially for two modules
The supervisor can set different measurement types	Pass
The supervisor only does measurements of modules before the given stop date	Pass
The supervisor creates a process for the daily loop that updates to the database	Pass
The supervisor creates a process for the update loop that updates to the database	Pass
The supervisor schedules measurements of modules where the stopdate is specified as null	Pass
The supervisor does not schedule any measurements if the module is disabled	Pass
The supervisor schedules a `set_load_mode` job that disables the OPET if it is disabled	Pass
The supervisor keeps trying to do measurements even if the OPET is disconnected	Pass
The supervisor is robust to missing or incorrect entries in the measurement configuration file	Pass, only `module_name` should exist
The supervisor is robust to syntax errors in the measurement configuration file	Pass

Test	Pass/Fail
Remove jobs if they are scheduled longer than five minutes ago	Pass
Stopped processes get restarted	Pass
The supervisor runs automatically on boot on the monitoring station PC	Pass
The supervisor is able to run on the monitoring station PC	Pass

All these tests are conducted on a PC of one of the team members unless specified otherwise. The supervisor passes almost all the tests. It is able to schedule measurements, perform them, and store the results in a CSV file. Additionally, the update loops for the database are started. In the case that processes die unexpectedly, they are restarted. Some additional tests are added to test the robustness of the system, as it is robuts to errors made by the user in the measurement configuration file. Unfortunately, a full stress test with 32 OPETs was not possible as there were only two OPETs available. Therefore, no validation can be found on whether it is able to measure the maximum of 32 OPETs at the same time. The limit is 32 since the OPET only has 32 different configurable addresses.

4.3. Firmware

Tests related to the firmware can be seen in Table 4.3. Additional information about each test can be seen in Section B.3.

Table 4.3: Tests to verify requirements and functionality for the firmware

Test	Pass/Fail
Monitor temperature with the MAX31856 with point measurement	Pass
Monitor temperature with the MAX31856 with curve measurement	Pass
Monitor the temperature of eight power dissipation devices	Partially for four devices
The output gets disabled due to the temperature rising above 30 °C	Pass
The output gets disabled due to the temperature rising above 70 °C	Pass
The output re-enables when dropping 5 degrees below the temperature limit	Pass
The output is disabled when no NTC is connected	Pass
The output is disabled if fewer ADCs respond than expected	Pass
Output is disabled if a communication error occurs with the one ADC	Pass
Temperature accuracy measurement	Pass, within 2 °C in steady state

The written firmware is compiled using Microchip Studio, which is an IDE used for AVR-type microcontrollers. The new firmware is flashed on the OPET using the Atmel-ICE programmer, which connects to the ISP pins on the OPET.

With the new firmware and the MAX31856 installed on the OPET, the user is able to receive the module temperature with point and curve measurements. The temperatures read from these tests were similar to the temperatures given by the onboard NTC of the OPET, validating that it works. Additionally, the measured temperature increased when the thermocouple was heated up by hand.

Unfortunately, it was not possible to monitor eight power dissipation devices at the same time. There were five boards available, of which only four correctly responded over I2C. The other board likely has a faulty or misaligned component on the PCB.

To verify whether the temperature sensor is accurate enough as set by requirement PER-4, a separate test with a calibrated temperature sensor was conducted. The firmware of the OPET is adjusted to only measure one ADC with the NTCALUG01A103J, such that a new temperature is given every 500 ms. The Yocto-PT100 temperature sensor is used in combination with a 4-wire PT100 probe. This reference sensor is capable of an accuracy of 0.03 °C at 0 °C [26]. Both sensors are mounted on a temperature-controlled hot plate with thermal isolation tape. This setup can be seen in Figure 4.1. The balancing board is the board that the other hardware subgroup has developed. For this test, only the ADC on this board is used to read the voltage.

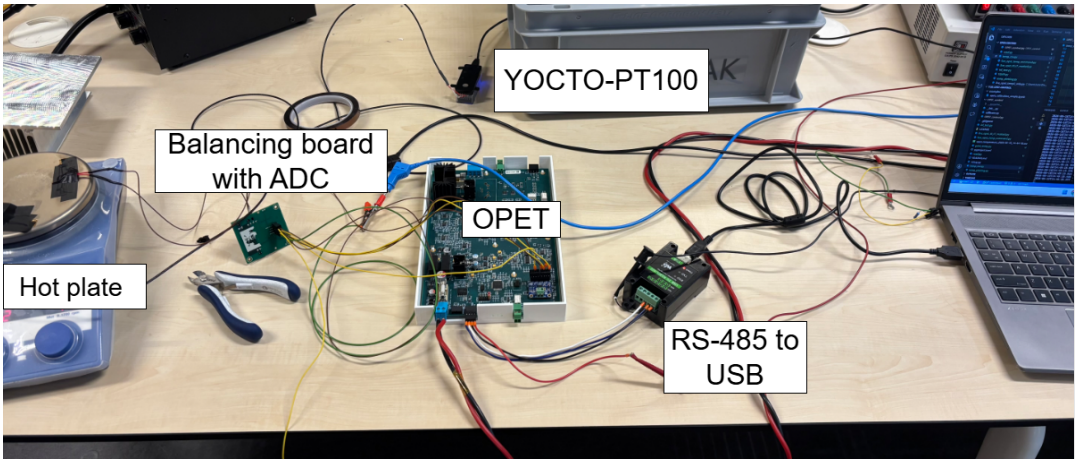

Figure 4.1: Test setup to test the accuracy of the temperature monitoring setup

Both sensors communicate their results to a laptop, and the results are stored in a CSV file. The temperature of the hotplate is increased four times. After increasing the temperature, some time was waited for both sensors to stabilize. The result of this measurement can be seen in Figure 4.2.

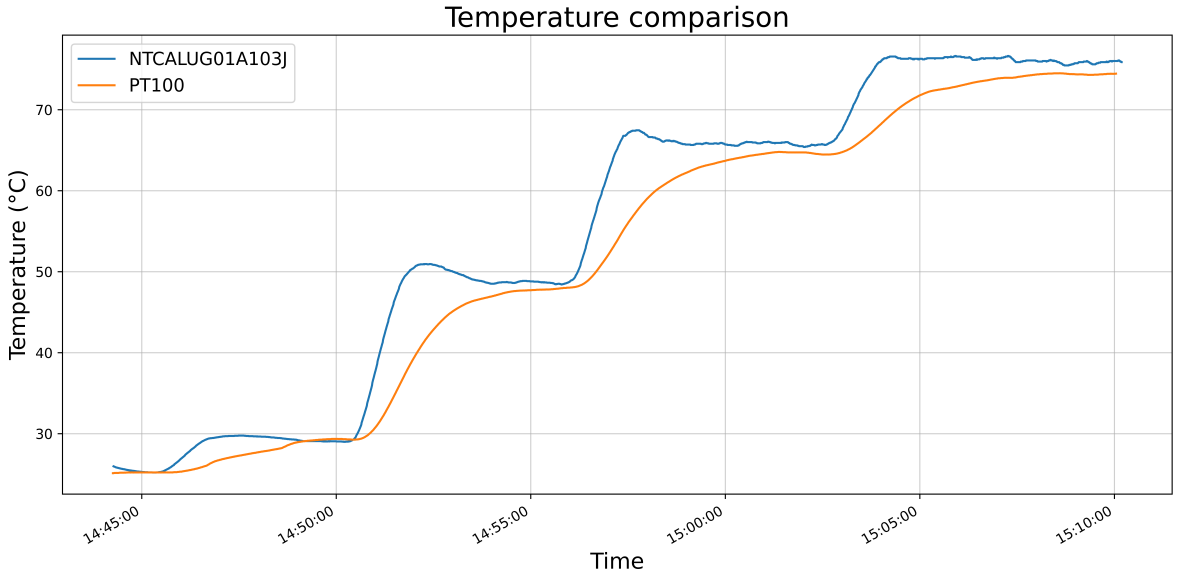

Figure 4.2: Measured temperature for both the PT100 and NTCALUG01A103J during the complete test

Each time the temperature of the hotplate increased, the NTCALUG01A103J heated up much faster, due to a different thermal response time. This could be due to the fact that the NTC thermistor is smaller than the PT100. The smaller footprint is easier to warm up. One noticeable observation is that the NTCALUG01A103J overshoots and then settles for a lower temperature. It is suspected that this is due to the heat plate used, as it might heat up a lot when a higher temperature is set and then settle a bit lower when the whole plate has reached that temperature. This happens before the PT100 reaches that temperature, so this theory could not be verified.

Additionally, the temperature difference ΔT can be calculated by the absolute difference between the temperature of the PT100 and the NTCALUG01A103J. This can be seen in Figure 4.3. From this figure, it can be verified that in steady state, ΔT does not exceed 2 °C, which means the system complies with PER-4. When both sensors are heating up, there is a big difference in temperature, which was expected due to the different sizes of the sensors.

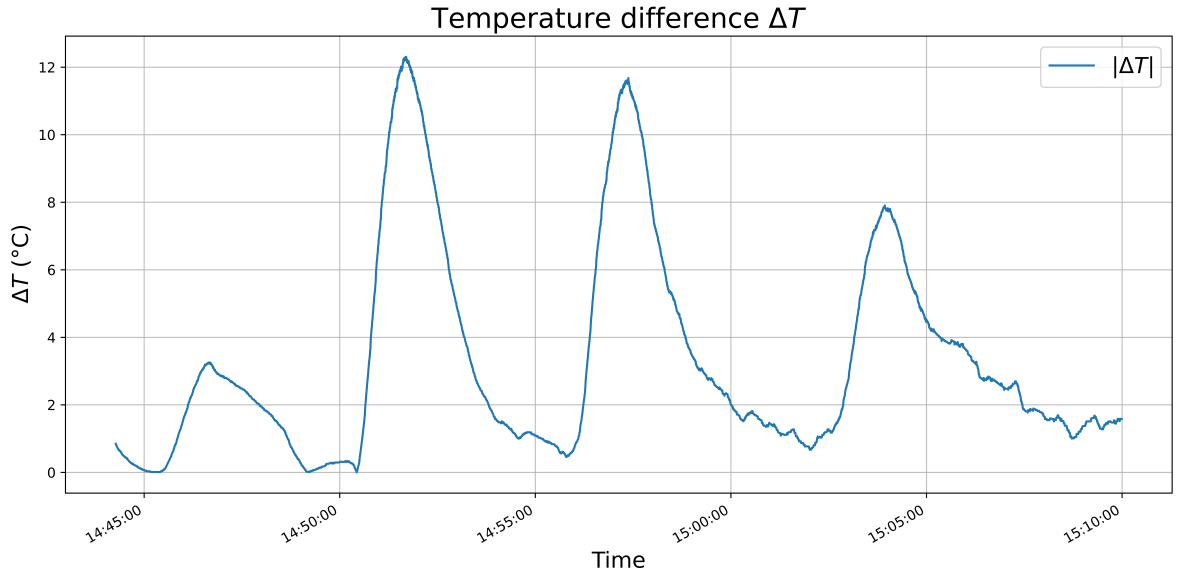

Figure 4.3: Temperature difference between the NTCALUG01A103J and the PT100 during the test

4.4. Concept demonstrator

The concept demonstrator will show the performance of the entire integrated system. This demonstrator has to contain data collection, storage, and retrieval to be in line with the requirements. To test this, it contains the following elements:

- 24 volts power supply to power the OPET
- 5 volts power supply for the RS-485 bus and bias voltage
- Solar module
- Lamp
- Database PC that runs the database and the `supervisor.py`
- High current OPET with the MAX31856 installed
- MOSFET mounted on a heatsink
- USB TO RS-485 converter
- T-Type thermocouple
- User PC that collects and plots measurement data

The setup, as seen in Figure 4.4, contains the components listed above. The first step is to connect the OPET to the 24-volt power supply to supply its power. The second step is to connect the 5-volt power supply to the RS-485 communication pins and to the bias power of the OPET. Additionally, the power dissipation MOSFET is connected to the OPET. The PV module is connected to the PV voltage and PV current ports of the OPET. Then, the OPET is connected to the RS-485 to USB converter, which connects to the computer that runs the supervisor. Finally, the thermocouple, which is connected to the MAX31856 sensor on the OPET, is attached to the back of the PV module. The database computer will run the database and store the measurements made by the OPET. This computer needs to be connected to Tailscale. The other computer running `live_data.py` also needs to be connected to the same Tailscale network, so that it can pull the data from the database.

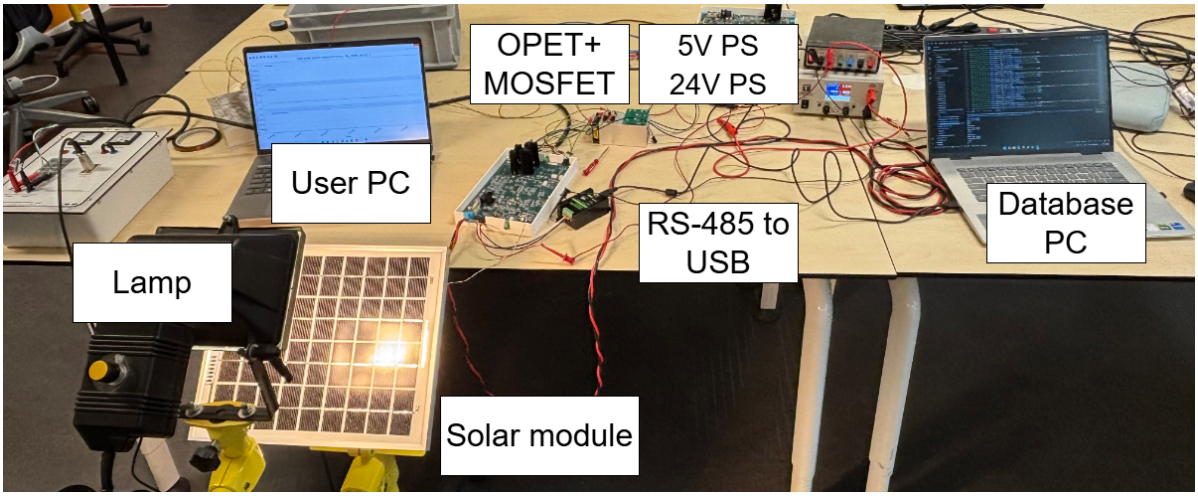

Figure 4.4: Testing setup of the concept demonstrator

To do the test, first turn on all power supplies. Then start the `supervisor.py` and then turn on `live_data.py`. In the measurement configuration file, the OPET is instructed to do maximum power point tracking. The live data will plot the point measurement with the current, voltage, and module temperature. Changing the light intensity on the solar module results in changing the voltage and current readings.

In Figure 4.5, the results of the demo can be seen. These plots are created on the user PC, which is connected to the same Tailscale subnetwork as the database PC, which takes the measurements. At first, the lamp is dimmer, and the voltage is slightly above 16 V. After setting the lamp to the brightest setting, the voltage rises to above 18 V. There is also a jump in the current, rising to almost 300 mA. This increase in voltage and current is expected since making the lamp brighter would result in a higher power output of the solar module. Additionally, due to the bright light, the temperature of the solar module gets increasingly hotter, which can be verified in the plot.

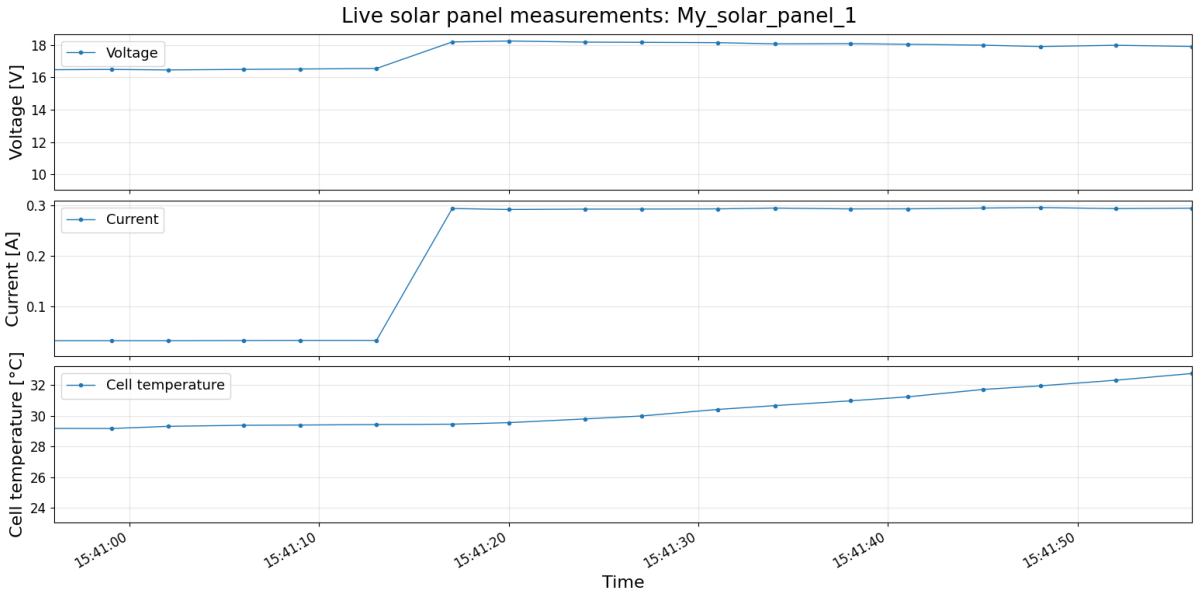

Figure 4.5: Measurement results of the voltage, current, and cell temperature received by the user PC

5 Discussion

In this chapter, the results presented in Chapter 4 are discussed and reflected. Section 5.1, 5.2, and 5.3 discuss the results of the database, while Section 5.4 and Section 5.5 discuss the supervisor and firmware. Finally, the performance of the concept demonstrator is discussed in Section 5.6.

5.1. Database tests

An important fact to note about the tests performed above is that they were all done on a PC laptop. The intent of the project was to get all the software installed on a server administered by TU Delft. However, there were some delays, so there was too little time to perform the tests on the server. The server passes most of the tests above, but the database on the server is not yet accessible over the network, because ICT permission is needed.

5.2. Updating database performance

As seen in the results, 2 different machines running the same program can have wildly different outcomes. The reason why the performance between 2 machines running the same application can be explained by the fact that one of the 2 had a higher clock speed than the other. This means that the performance of the system is heavily dependent on the specs of the server. The reason why this is, is because there is a sequential operation that completely relies on clock speed. The insertion of the point and curve measurement into the database is a sequential operation (Section D.1.3). This takes up nearly all of the add data operation. There may be a way to upgrade this, using `mogrify()` [27]. How significant the gain is and whether that really solves the problem remains to be tested. The situation that was presented is not completely realistic, because the CSV file that needs to be uploaded will never be that large. Besides, the CSV file grows throughout the day, so for the largest part of the day, the performance will be good. Only at the end of the day, it might get quite slow.

5.3. Weather ID

Although the design has been thoroughly thought through, it is not without imperfection. One of the biggest design trade-offs lies within the weather ID synchronisation. In this program, `None` is assigned if it is not ready to be given a weather ID. When it does get assigned an ID, but no measurement was within range, it gets a 0. This distinction is great and allows the system to check efficiently only for groups that have not yet been assigned a weather ID. This was done to make the program run as efficiently as possible, because it needs to query less information. Otherwise, the query would get slower as time goes on, because measurements with ID 0 would accumulate. However, there is a downside to this. When assigning the weather ID, there was no suitable weather measurement, but later on, a weather measurement was retroactively uploaded. The weather ID will not get updated again to the newly best-fitting.

5.4. Supervisor reliability

In the results, only a test with two OPETs at the same time is performed. However, the system should be able to handle up to 32 OPETs. Ideally, a test with 32 OPETs should have been done to see whether the supervisor is able to handle it. Also, it would be insightful to see if the RS-485 bus is able to communicate with 32 OPETs at the same time. If this does not work, an extra bus or buses should be added. Now, no decisive conclusion can be taken on whether this setup is able to handle more than two solar modules in the new monitoring station. However, the tests from the supervisor are promising as they are able to fulfil the other requirements and tests.

While testing the auto start of the supervisor, one problem was encountered. The Python script will be able to run when the user logs in. However, if the user is logged out due to an update, for example, an admin should first log in to the admin account before the supervisor continues. So after the shutdown, the admin should log in to get the supervisor started.

5.5. Firmware tests

The new firmware could not be tested with the full capacity of eight boards. However, since the system is validated with four devices and only one device is measured each cycle, it is expected to work with eight. One downside is that the update speed is not very high for each power dissipation device. However, this is not a problem with the reaction speed of disabling the output, since the alert pin is checked each time one temperature is updated. Alternatively, it might be possible to double the update by measuring the PV module temperature and one power dissipation device each cycle. This can be achieved by increasing the communication speed. This solution has not been tested.

The accuracy test of the temperature monitoring device shows that it will be able to monitor the temperature within the set requirements of 5 °C. However, the test shows some overshoot, but it is unclear why it happens. The overshoot does not have a bigger difference than 3 degrees compared to the stable temperature. So this temperature setup is considered good enough for this application.

5.6. Concept demonstrator

Although the concept demonstrator shows the main functionalities of the new design, there are some caveats. The database runs on one of the team members' laptop, as it is infeasible to bring the server desktop to the final exam. The server desktop still has to get some settings changed for it to be remotely accessible. Performance might differ per system as discussed in Section 5.2. Additionally, only one module is tested, whereas in the real scenario, multiple modules will be used to take measurements.

Additionally, the requirements `REL-1` and `REL-2` are not explicitly tested or verified. It is not possible to test the exact reliability of the system over a longer period of time. However, during the design, these requirements are kept in mind to keep the system robust and reliable. In addition, some tests are created to test certain vulnerabilities, such as human-made errors. Moreover, sufficient documentation about the system must be present to keep the system maintainable.

6 Conclusion

6.1. Conclusion

Due to the rising interest in solar modules, the TU Delft wants to establish an outdoor testing ground to measure the performance of new solar cell technologies. This bachelor's graduation project discusses the design of this new testing ground using an open-source PV measurement tool called OPET.

A program of requirements was developed to identify specific goals based on the needs of the stakeholders. To verify the performance and requirements of the design, pass/fail tests were created. These tests were based on checking whether the requirements are met and to check various special cases to test the reliability of the system. With these tests, the basic operation of the system can be verified.

A test setup has been created to test the possibilities of the new database. The database is remotely accessible over the network for authorized users, can retroactively add data, and enables users to export information about their solar module. When the weather database starts collecting the data, the database should retrieve this information. Any possible errors are logged and emailed to the server administrator.

Additionally, a supervisor has been developed that starts the various processes for measuring and storing PV data. It is able to start the different Python processes and schedule PV measurements. The user is able to configure the measurements by setting information such as the interval between measurements and the measurement type, as well as various metadata about the module. The supervisor is able to pass every test with the available OPETs. However, a full test with 32 OPETs was not possible.

The firmware of the OPET has been adjusted to measure the PV temperature with a new sensor. In addition, the OPET is now able to monitor the temperature of up to eight power dissipation devices. These devices are developed by the other subgroup of this project. The output of the solar module is disabled when the temperature of the power dissipation MOSFET gets too high. The measured temperature difference of the new temperature monitoring stayed within 2 °C in steady state compared to a reference sensor. Additionally, all the available temperature information can be monitored with a new communication command.

The concept demonstrator is able to demonstrate functionalities of the new measurement setup. One laptop runs both the database and the supervisor, which schedules the measurements. The supervisor communicates with the OPET to make the measurements. Another laptop is able to connect to this database and retrieve the new measurement data, with the voltage and current readings, as well as the temperature of the module with the new temperature sensor. However, the reliability of the system is yet to be explicitly tested.

6.2. Future work

Before the developed setup can be implemented at the outdoor monitoring station, multiple steps need to be taken. Access to the server PC has been granted, and the required programs are installed. However, the server has not been set up on the correct network due to some ICT permissions. That is why, for testing, Tailscale has been used. The server is also not remotely accessible yet, because some configuration files still need to be changed by ICT. Additionally, the database does not retrieve live information from the weather database, because the weather database is currently not collecting any data. The database can currently only send emails via a Gmail account, because the functional mailbox is yet to be made ready for use as an SMTP server by ICT.

The data insertion of the database can take increasingly longer as the number of rows that need to be inserted grows. Although this issue is not expected to be very significant for the operation of the database, it may only introduce some delays. So it may be worth looking into faster possibilities to insert the data into the database.

Before installing the new supervisor, it should be checked whether it is able to execute measurements of 32 OPETs at the same time. Otherwise, an extra bus needs to be implemented.

A faster update time of the temperature monitoring can be achieved in an improved design when not all devices are installed. However, currently, an update speed of 8 seconds is deemed fast enough since the alert pin is checked every second.

To enhance the user experience an interface should be made. This interface can be an application from which the database is accessed or a website from which the data can be gathered. This would make it easier to use the database for people who do not know Python or SQL.

References

[1] Delft University of Technology, *Dutch pv portal*, Figure obtained from TU Delft Dutch PV Portal. Accessed: 2026-04-21, 2026. [Online]. Available: https://www.tudelft.nl/ewi/over-de-faculteit/afdelingen/electrical-sustainable-energy/photovoltaic-materials-and-devices/dutch-pv-portal.

[2] M. Crippa, D. Guizzardi, F. Pagani, M. Banja, M. Muntean, et al., "Ghg emissions of all world countries – 2025 report," Publications Office of the European Union, Luxembourg, JRC143227, 2025. DOI: 10.2760/9816914.

[3] International Energy Agency (IEA), "Renewables 2022," International Energy Agency, Paris, 2022, Licence: CC BY 4.0. [Online]. Available: https://www.iea.org/reports/renewables-2022.

[4] National Laboratory of the Rockies (NLR), *Best research-cell efficiency chart*, 2026. [Online]. Available: https://www.nrel.gov/pv/cell-efficiency.

[5] M. Schachinger. "Market analysis april 2026 - solar module price increase continues, for now." Accessed: 2026-04-23. [Online]. Available: https://www.pvxchange.com/Market-Analysis-April-2026-Solar-module-price-increase-continues-for-now.

[6] International Energy Agency (IEA), "Solar pv global supply chains," International Energy Agency, Paris, 2022, Licence: CC BY 4.0. [Online]. Available: https://www.iea.org/reports/solar-pv-global-supply-chains.

[7] *Terrestrial photovoltaic (pv) modules - design qualification and type approval - part 1: Test requirements*, IEC 61215-1:2021, International Electrotechnical Commission.

[8] NatLabRockies, *Opet hardware: Open-source photovoltaic electrical tool hardware repository*, GitHub repository. Accessed: 2026-04-21, 2026. [Online]. Available: https://github.com/NatLabRockies/opet-hardware.

[9] NatLabRockies, *Opet control software (cleanup branch)*, GitHub repository. Accessed: 2026-04-21, 2026. [Online]. Available: https://github.com/NatLabRockies/opet-control/tree/cleanup.

[10] NatLabRockies, *Opet firmware*, GitHub repository. Accessed: 2026-04-21, 2026. [Online]. Available: https://github.com/NatLabRockies/opet-firmware.

[11] National Laboratory of the Rockies, *National laboratory of the rockies*, Accessed: 2026-04-21, 2026. [Online]. Available: https://www.nlr.gov/.

[12] G. van Rossum, B. Warsaw, and A. Coghlan, *PEP 8 – Style Guide for Python Code: Naming Conventions*, Accessed: 2026-05-29, 2001. [Online]. Available: https://peps.python.org/pep-0008/#naming-conventions.

[13] DuraMAT, *PV Modeling Glossary*, Version 0.1.2. Accessed: 2026-05-29. [Online]. Available: https://duramat.github.io/pv-terms/index.html.

[14] Kamerplant71, *opet-firmware-TUD*, Accessed: 2026-05-29. [Online]. Available: https://github.com/Kamerplant71/opet-firmware-TUD.

[15] Kamerplant71, *TUD-opet-control*, Accessed: 2026-05-29. [Online]. Available: https://github.com/Kamerplant71/TUD-opet-control.

[16] Kamerplant71, BonkerGod, *TUD-PV-monitoring-tool*, Accessed: 2026-05-29. [Online]. Available: https://github.com/Kamerplant71/TUD-PV-monitoring-tool.

[17] Tailscale Inc. "Tailscale. "[Online]. Available: https://tailscale.com/.

[18] The PostgreSQL Global Development Group, *Postgresql 18.4 documentation*, The PostgreSQL Global Development Group. [Online]. Available: https://www.postgresql.org/docs/18/index.html.

[19] The PostgreSQL Global Development Group. "About postgresql," The PostgreSQL Global Development Group. [Online]. Available: https://www.postgresql.org/about/.

[20] GeeksforGeeks. “Difference between mysql and postgresql,” GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/mysql/difference-between-mysql-and-postgresql/.

[21] Analog Devices, *MAX31856 Precision Thermocouple to Digital Converter with Linearization Data Sheet*, Accessed: 2026-05-29. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/max31856.pdf.

[22] Microchip Technology, *ATmega1284 Data Sheet*, Accessed: 2026-05-29. [Online]. Available: https://ww1.microchip.com/downloads/en/devicedoc/atmel-42718-atmega1284_datasheet.pdf.

[23] Vishay, *Selecting NTC Thermistors*, Accessed: 2026-05-29. [Online]. Available: https://www.vishay.com/docs/33001/seltherm.pdf.

[24] Vishay, *NTC Thermistors, Standard Lug Sensors, NTCALUG01A Series*, Accessed: 2026-05-29. [Online]. Available: https://docs.rs-online.com/a717/0900766b8167f03d.pdf.

[25] Texas Instruments, *ADC121C021/ADC121C021Q/ADC121C027 I 2C-Compatible, 12-Bit Analog-to-Digital Converter with Alert Function*, Accessed: 2026-05-29. [Online]. Available: https://www.ti.com/lit/ds/symlink/adc121c021.pdf.

[26] Yoctopuce. “Yocto-pt100.” Accessed: 2026-06-17. [Online]. Available: https://www.yoctopuce.com/EN/products/usb-environmental-sensors/yocto-pt100.

[27] GeeksforGeeks. “Python psycopg2 - insert multiple rows with one query,” GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/python-psycopg2-insert-multiple-rows-with-one-query/.

A
Ethical Implications

A.1. Societal impacts

Our product, the database with the OPET collection system, has various stakeholders. There are 2 primary stakeholders. The primary stakeholder is the PVMD group; they are the ones who are responsible for the product that we produce. They will be responsible for the good operation, maintenance, and data collection. The other important stakeholders are the researchers who are going to be using our end product. They will benefit from easy and quick data gathering. They will be able to, regardless of the module size, test their new solar module. Two secondary stakeholders are the TU Delft and the PV community. TU Delft provides the subsidies and the location. The PV community might benefit from the improved solar modules that are being developed.

A.2. Ethical aspects of development

The most relevant ethical aspects of this project are the data collection and the environmental impact. The users of our benchtop might be vulnerable to some private data collection. The names and email addresses of the researcher are stored together with the performance and characteristics of their module. Additionally, the temperature monitoring of the new power dissipation device can cause environmental damage. If it lets the system get too hot, fire hazards may emerge, which could lead to damage to the building and possibly the surroundings and endanger employees inside the building. The system should monitor the heat carefully and disconnect the output in case of an over-temperature event.

A.3. Ethical evaluation

The privacy issue raised some ethical concerns for the team. Because the name of the user and their email address are collected. This was done so that anyone can find out whose module it was, and that possible errors can be sent to the owner. However, there is a way out. We have given people the ability to remove their username and user email from the database. This gives people an opt-out option if they are really worried about their data privacy. However, this database is currently only aimed for in-house use, and thus these names and emails will not be spread far and wide. So the real risk of their data privacy being breached is small. The possibility of environmental damage also raised some concern. The output will not only turn off if the measured temperature is too high, but it will also disconnect the output when a communication error occurs. There might be a communication error even when the temperature is fine. This is a trade-off between safety and performance, where we thought safety is more important than always getting measurement data.

A.4. Reflection on design choices

One concern was that researchers' names and emails were being collected. However, we found the ease of finding out whose PV module it was and the error detection to be of more worth. This way, researchers are directly notified in case the system fails and do not need to rely on the server administrator. For safety reasons, an extra error status was added to detect communication errors with the temperature monitoring device. Therefore, taking communication errors into account results in a safer system. Otherwise, the temperature of the dissipation device might rise without supervision.

A.5. Looking-forward

If our concept were used in a real-world application, the ethical implications would not differ that much. However, when the database collects the data of various organizations, companies, and/or customers in a centralized way, there would be a significant risk to this product. Companies might be testing out

their most advanced PV modules that are yet to go to market, but because they use our equipment, we would already know the results of their measurements. So to prevent this, the data should never be allowed to centralize and it should always be stored at the organization that uses the database. For the heat monitoring device, more tests should be conducted to test if it responds as expected in different environments. It would be valuable to know if there are any unexpected situations where the system might break, which can lead to temperature/fire hazards. For example, the system is not tested with the maximum number of dissipation devices possible.

B
Tests to verify requirements

B.1. Database tests

Table B.1: Tests to verify requirements and functionality for the database

Test	Criteria	How	Pass/Fail
An authenticated user is able to retrieve measurement data	The user gets a .csv file by running a Python script on their PC	Test the function by applying the filter and checking for the correct output. Verify this by controlling the contents of the database. Make sure they fill in the correct account info.	Pass
User is able to connect remotely to the database	The user receives the message that they have successfully connected to the database	Go to a different computer and check whether the connection is established	Pass
The database is able to connect to the weather database	The database sends the message that it has connected to the weather database	Run the MySQL connector and see if it is able to connect to it.	Pass
Curve and measurement data are being added	When the print_table() shows all the new entries. The OPET data gets added every 10 seconds.	Keep collecting data and adding at the same time. Print the entry count. This should grow in steps of 1 or 2. At the end, print or download the entire table and verify	Pass
Live OPET measurement collected via update loop	Data gets added to the database when running the update loop	Run the supervisor and connect the OPETs, and check whether new entries are being created.	Pass
Live weather data collected via update loop	Data gets added to the database when asking for it from the weather database	Run the supervisor and make sure the database gets connected to the weather database. Check if new entries get created.	Fail, the system is able to connect to the weather database. It, however, does not make any new measurements.
Adding the past OPET data with a lookback time retroactively	When the past data gets listed when calling the print_table statement	Create an empty table and run the past data upload, and check whether the data before the current day shows up	Pass
Adding the past weather data with a lookback time retroactively	When the past data gets listed when calling the print_table statement	Create an empty table and run the past data upload, and check whether the data before the current day shows up	Pass

Test	Criteria	How	Pass/Fail
Adding the module data to the database	The module data is shown in the table	After adding the data to the JSON, run the `add\_data()` script and see whether the data has been added to the modules table.	Pass
Add module data with one element missing	The module data is shown in the table	Create an empty table and add a module; however, leave one element empty	Pass, except for the 'module_name'
Adding the weather data to the database	Check whether data from years ago can be added to the database by changing the start date.	Run the program and print the tables before and after, and see whether the data gets added	Partially, this data is from 2025 and before.
Sending an email via SMTP Python	Receive an email	Send a message and see if it arrives	Partially, currently only via Gmail server
Detect an error when no data is being received by the database	Send an email to the admins when the database is not collecting any data at all	No measurement data collected in the past 24 hours, and receive an email	Pass
Detect the number of errors for a day of measurements per day	Send an email to the admins and the module user when there are errors in the OPET measurements	Have a measurement in the past 24 hours and insert a 'status_integer' error	Pass
Detect an error when data is not being received from a single module	If all measurements for some solar modules work, but for others it does not. Send an email about which do not work	Have a module collect data and another not. See if an email is sent	Pass
Retroactively sync the weather ID of the weather data to the measurements	After trying to sync the data, the `weather_id` should be synced when there was a fit within 5 minutes of each other.	Make a synthetic weather measurement and check if it assigns the ID correctly to the PV data	Pass
Module data can be updated only for tracer, username, user_email	If module_name, area, technology, and manufacturer are the same, it will change the username, email, and tracer	Create an artificial module and see if the tracer, username, and email can be changed	Pass
Module data created with a duplicate name will not get added, and the user gets notified via the log	If a duplicate name is used, it will be logged in the log file	Try to add a duplicate name and see if it gets logged	Pass
If the email address is invalid, it gets logged	If the email address is invalid, it gets logged	Enter an invalid email address and see if it gets caught	Pass
Other measurements still get added when the weather data is not connected	OPET data in the database still gets updated when the weather database is not connected	Remove connection with the weather DB and run the update_loop	Pass

Test	Criteria	How	Pass/Fail
Checking if the database has collected data by receiving the most recent measurement	Get a measurement that was a couple of minutes ago	Try to connect to the database and receive the last measurement. If the database cannot connect, it is offline; if the measurement is old, there is something wrong	Pass

B.2. Supervisor

Table B.2: Tests to verify requirements and functionality for the supervisor

Test	Criteria	How	Pass/Fail
Schedule, do, and save measurements	The supervisor is able to store point and curve data in a CSV file of 32 OPETS	Daisy chain 32 OPETs together and start the supervisor	Partially, with two modules
The system is able to do at least one point measurement every 30 seconds and one curve measurement every 5 minutes for each module within the performance measures set in PER-2, PER-3	The supervisor is able to store point and curve data in a CSV file of 32 OPETS without exceeding the delays set in PER-2, PER-3	Daisy chain 32 OPETs together and start the supervisor. Set in the config to do a point and curve measurement every 30 and 300 seconds for each module.	Partially, with two modules
The supervisor can set different measurement types	The supervisor is able to set the OPET to Voc, Isc, and MPPT. No errors should be present in the log file.	Run the supervisor and set in the configuration the different measurement types	Pass
The supervisor only does measurements of modules before the given stopdate	The supervisor does not schedule any measurements if the stopdate is in the past. It does schedule measurements if the stopdate is set in the future.	Run the supervisor and set in the configuration the stopdate to a date in the past	Pass
The supervisor creates a process for the daily loop that updates to the database	The daily loop process in the supervisor is able to retroactively adding measurements and reporting error detection	Set the time of the daily loop to the current hour and run the supervisor, while forcing an error	Pass
The supervisor creates a process for the update loop that updates the database	The update loop process in the supervisor is able to update the measurements into the database every 10 seconds	Run the supervisor and see if the data gets added to the database	Pass
The supervisor schedules measurements of modules where the stopdate is specified as null	The supervisor schedules the measurements of a module where the stopdate is specified as null	Run the supervisor and set in the configuration the stopdate to null	Pass
The supervisor does not schedule any measurements if the module is disabled	The supervisor does not schedule any measurements of the module which is disabled	Run the supervisor and disable the module in the configuration	Pass

Test	Criteria	How	Pass/Fail
The supervisor schedules a `set_load_mode` job that disables the OPET if it is disabled	The output of the OPET is disabled once the configuration file updates a module to disabled	Run the supervisor and disable the module in the configuration	Pass
The supervisor keeps trying to do measurements even if the OPET is disconnected	The supervisor does not crash due to a timeout error and keeps scheduling measurements. Once the OPET is reconnected, it saves the new measurements in the CSV file.	Run the supervisor without OPET connected. Once the measurement time is exceeded reconnect the OPET.	Pass
The supervisor is robust to missing or incorrect entries in the measurement configuration file	The supervisor does not crash due to a key missing in the measurement configuration file	Run the supervisor and remove one by one entries of the measurement configuration file	Pass, only module_name should exist
Supervisor is robust to syntax errors in the measurement configuration file	The supervisor will keep using an older version of the measurement configuration file when a syntax error is introduced.	Run the supervisor, and while it's running, add a syntax error	Pass
Remove jobs if they are scheduled longer than five minutes ago	The supervisor removes jobs that are older than the specified maximum time a job is allowed to exist.	Run the supervisor without OPET connected. The supervisor will schedule measurements, but they won't be completed. The maximum time for a job can be set to 30 seconds.	Pass
Autorun on restart	Supervisor automatically restarts upon restart	Restart the PC and check if the system correctly restarts	Pass
Stopped processes get restarted	The supervisor is able to restart each process that stopped due to an error	Run the supervisor and force an error in one of the processes, and temporarily remove the try, except statement to let the process die.	Pass
The supervisor is able to run on the monitoring station PC	The supervisor is able to run normally without errors. Additionally, new dummy measurements get stored in CSV files and consequently in the database.	Connect the OPET to the monitoring station PC and start the supervisor.	Pass

B.3. Firmware

Table B.3: Tests to verify requirements and functionality for the firmware

Test	Criteria	How	Pass/Fail
Monitor temperature with the MAX31856 with point measurement	The host PC is able to receive the temperature from the MAX31856 with a point measurement.	Use a host PC connected to an OPET with the MAX31856 and thermocouple installed. Send a `.sample` command from the host PC to one OPET.	Pass
Monitor temperature with the MAX31856 with curve measurement	The host PC is able to receive the temperature from the MAX31856 with a curve measurement.	Use a host PC connected to an OPET with the MAX31856 and a thermocouple installed. The host PC reads the result of an IV curve.	Pass

Test	Criteria	How	Pass/Fail
Monitor the temperature of eight power dissipation devices	Host is able to receive the temperature of eight power dissipation devices with the TEMP? command.	Use a host PC that sends a temperature command to one OPET with eight power dissipation devices with ADCs installed.	Partially for four devices
The output gets disabled due to the temperature rising above 30 °C	The OPET turns on the red error LED, sets the over-temperature and alert error bit, and disables the output.	Turn on the OPET while warming up the NTC with a soldering iron. The temperature limit is set to 30 °C. Temperature and status bits are monitored with a host PC.	Pass
The output gets disabled due to the temperature rising above 70 °C	The OPET turns on the red error LED, sets the over-temperature and alert error bit, and disables the output.	Turn on the OPET, which is connected to a solar module. The temperature limit is set to 70 °C. The MOSFET is heated up with a soldering iron. Temperature and status bits are monitored with a host PC.	Pass
The output re-enables when dropping 5 degrees below the limit	The OPET turns off the red error LED, clears the over-temperature status bits, and turns on the output after it measures below 65 degrees	Let the thermal limit be reached as in the test above. Afterwards, let it cool down and see whether the output turns back on. Temperature and status bits are monitored with a host PC.	Pass
The output is disabled when no NTC is connected	The OPET turns on the red error LED, sets the ADC offline status bit, and disables the output	Don't connect an NTC to the ADC. Check the error bits of the OPET with a host PC. Status bits are monitored with a host PC.	Pass
The output is disabled if fewer ADCs respond than expected	The OPET turns on the red error LED, sets the ADC offline status bit, and disables the output.	Turn on the OPET output without the power dissipation device attached. Status bits are monitored with a host PC. The test is repeated with one device installed, while the OPET is configured to measure two. Status bits are monitored with a host PC.	Pass
Output is disabled if a communication error occurs with the one ADC	The OPET turns on the red error LED, sets the ADC offline status bit, and shuts off the output.	Turn on the OPET output with the power dissipation device attached. During the operation, cut a communication wire. Status bits are monitored with a host PC.	Pass
Temperature accuracy measurement	At steady state, the difference between the temperatures do not deviate more than 5 °C as set in PER-4	The NTC is mounted on a hot plate with a reference temperature sensor. The hot plate temperature is set between 20 and 75 °C	Pass, within 2 °C in steady state

C Firmware Source Code

C.1. Setup MAX31856

Code Listing C.1: Setup function for the MAX31856

```
void TEMP_MAX31856_Setup(){
        uint8_t ADC_CFG; //Address
        uint8_t ADC_SET; //Data

        // init communication (set SPI interface)
        SETBIT(SPCR,SPR0);
        SETBIT(SPCR,SPR1);      // clock is now fck/128
        SETBIT(SPCR, CPHA);  // rising edge output


        // write configuration0
        ADC_CFG = 0b10000000; // write, register config0
        ADC_SET = 0b00000010; // no fault detection, clear faults
        #ifdef Line_Freq_50 // adjust config byte if 50Hz line frequency
                ADC_SET = ADC_SET + 1;
        #endif /* Line_Freq_50 */

        //Start communication
        CLR__EXP_DIO_1;  // enable chip select
        SPDR = ADC_CFG;
        while(!(SPSR & BIT(SPIF))){ //Wait till register is sent
        }
        SPDR = ADC_SET;
        while(!(SPSR & BIT(SPIF))){//Wait till data is sent
        }
        SET__EXP_DIO_1;  // disable chip select


        // write configuration1
        ADC_CFG = 0b10000001; // write, register config1
        ADC_SET = 0b01110111; // Average over 16 samples, T-type ADJUST FOR OWN TYPE
            THERMOLCOUPLE
        //Start communication
        CLR__EXP_DIO_1;  // enable chip select
        SPDR = ADC_CFG;
        while(!(SPSR & BIT(SPIF))){ //Wait till register is sent
        }
        SPDR = ADC_SET;
        while(!(SPSR & BIT(SPIF))){//Wait till data is sent
        }
        SET__EXP_DIO_1;  // disable chip select


        // Enable auto conversion
        ADC_CFG = 0b10000000; // write, register config0
        ADC_SET = 0b10000000; // start autoconversion
        #ifdef Line_Freq_50 // adjust config byte if 50Hz line frequency
                ADC_SET = ADC_SET + 1;
        #endif /* Line_Freq_50 */

        //Start communication
        CLR__EXP_DIO_1;  // enable chip select
        SPDR = ADC_CFG;
        while(!(SPSR & BIT(SPIF))){ //Wait till register is sent
        }
        SPDR = ADC_SET;
```

```
        while(!(SPSR & BIT(SPIF))){//Wait till data is sent
        }
        SET__EXP_DIO_1;  // disable chip select


        // finish communication to device (reset SPI settings)
        CLRBIT(SPCR, CPHA);  // falling edge output main ADC
        CLRBIT(SPCR,SPR1);      // clock fck/16
}
```

C.2. Measure MAX31856

Code Listing C.2: Measure function for the MAX31856

```
//Measurement for MAX31856, returns temperature in Celcius as float
float TEMP_MAX31856_Measure(){
        float Temperature;
        uint8_t ADC_CFG;
    uint8_t LTCBH;
    uint8_t LTCBM;
    uint8_t LTCBL;
    int32_t raw;

        // start up the SPI,
        SETBIT(SPCR,SPR0);
        SETBIT(SPCR,SPR1);      // clock is now fck/128
        SETBIT(SPCR, CPHA);  // rising edge output


        //Start reading bytes
        ADC_CFG = 0x0C; // read, LTC_BH

        CLR__EXP_DIO_1; // enable chip select
        SPDR = ADC_CFG;
        while(!(SPSR & BIT(SPIF))){
        }
        SPDR = 0b11111111; //Send dummy byte
        while(!(SPSR & BIT(SPIF))){
        }
        LTCBH = SPDR;
        // read next byte
        SPDR = 0b11111111;
        while(!(SPSR & BIT(SPIF))){
        }
        LTCBM = SPDR; //LTC_BM
        // read next byte
        SPDR = 0b11111111;
        while(!(SPSR & BIT(SPIF))){
        }
        LTCBL = SPDR; //LTC_BL

        // finish up SPI, put back to normal
        SET__EXP_DIO_1; // disable chip select
        CLRBIT(SPCR, CPHA);  // falling edge output main ADC
        //CLRBIT(SPCR, CPOL); // normal low clock for main ADC
        CLRBIT(SPCR,SPR1);      // clock fck/16

        //Convert to temperature
        raw = ((int32_t)LTCBH << 16 ) | ((int32_t)LTCBM << 8 ) | LTCBL;
        raw >>=5; //Remove 5 lowest bits, temperature is only 19 bits including 1 sign bit

        //If signed
        if (raw & 0x40000) {
                raw |= 0xFFF80000;
        }

        Temperature = raw * 0.0078125;
        return Temperature;
}
```

C.3. Power dissipation device

Code Listing C.3: Source file including all functions used for the temperature monitoring

```
/*
        ----------------------------------------
        MPPT PCB MCU source
        ----------------------------------------
        Made by:        Justen van Koppen
        Version:        1.15
        Date:           20.05.2026
        ----------------------------------------
        ATMega 1284 Micro controller support
        ----------------------------------------
        Temperature monitoring C-Code FILE
        ========================================
*/
//=================================================================================================
// INCLUDE Header

#include "MPPT_PCB_MCU__Main.h"
#include "MPPT_PCB_MCU__Com.h"
#include "MPPT_PCB_MCU__IO.h"
#include "MPPT_PCB_MCU__EROM.h"
#include "MPPT_PCB_MCU__LOAD_CTR.h"
#include "MPPT_PCB_MCU__Range.h"
#include "MPPT_PCB_MCU__PI_CTR.h"
#include "MPPT_PCB_MCU__IV_Trans.h"
#include "MPPT_PCB_MCU__TEMP.h"

//=================================================================================================
// VARIABLES and STRUCTURES

const uint8_t ADDRESSES_ADC121C021[8]= [0x50, 0x51, 0x52, 0x54, 0x55, 0x56, 0x58, 0x59];

volatile float AI_HEAT_NTC_Temp[8];
volatile uint8_t AI_HEAT_ADC_Present_Mask;
volatile uint8_t AI_HEAT_ADC_Fault_Mask;


//=================================================================================================
// EEPROM VARIABLES

//=================================================================================================
// FUNCTIONS
//=================================================================================================
//-------------------------------------------------------------------------------------------------

// Try to setup all adc's
void Temp_monitoring_Setup() {

    AI_HEAT_ADC_Present_Mask = 0;
        AI_HEAT_ADC_Fault_Mask = 0;

    for (uint8_t i = 0; i < 8; i++) {
        if (ADC121C021_Is_Present(ADDRESSES_ADC121C021[i])) {
            AI_HEAT_ADC_Present_Mask |= (1 << i);
            ADC121C021_Setup(ADDRESSES_ADC121C021[i]);
        }

                AI_HEAT_NTC_Temp[i] = -9.9;

    }
}


// Read temperature of all ADC's
void Read_Temp_ADC121C021(uint8_t i){

        uint16_t raw;

        if (!(AI_HEAT_ADC_Present_Mask & (1 << i))){ //Check if ith device is present
```

```
                return;
        }

    if (ADC121C021_ReadRaw(i, &raw)) {
        AI_HEAT_NTC_Temp[i] = NTC_RawToTemp(raw);
                AI_HEAT_ADC_Fault_Mask &= ~(1 << i);
    }
    else {
        AI_HEAT_ADC_Fault_Mask |= (1 << i); //Set fault bit if communication fails
    }
}



// Setup ADC with correct values in registers
// I2C clock already configured in DIOExp_config_init as 111kHz
void ADC121C021_Setup(uint8_t address){

        ADC121C021_Set_Config(address);
        ADC121C021_Set_Vlow(address);
        ADC121C021_Set_Vhyst(address);

}


// Reads the 12 bit result of ADC and gives an success rate as output
uint8_t ADC121C021_ReadRaw(uint8_t i, uint16_t *raw_out){
    uint8_t address = ADDRESSES_ADC121C021[i];

    uint8_t msb = 0;
    uint8_t lsb = 0;

    // Send start
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_ReadRaw_Error;
    if ((TWSR & 0xF8) != START) goto ADC121C021_ReadRaw_Error;
    _delay_us(IC2_COM_DELAY_us);

    // Send address write
    TWDR = (address << 1);
    TWCR = (1 << TWINT) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_ReadRaw_Error;
    if ((TWSR & 0xF8) != MT_SLA_ACK) goto ADC121C021_ReadRaw_Error;
    _delay_us(IC2_COM_DELAY_us);

    // Send register
    TWDR = ADC121C021_REG_CONV_RESULT;
    TWCR = (1 << TWINT) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_ReadRaw_Error;
    if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_ReadRaw_Error;
    _delay_us(IC2_COM_DELAY_us);

    // Repeated start
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_ReadRaw_Error;
    if ((TWSR & 0xF8) != REPEATED_START) goto ADC121C021_ReadRaw_Error;
    _delay_us(IC2_COM_DELAY_us);

    // Send address read
    TWDR = (address << 1) | 1;
    TWCR = (1 << TWINT) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_ReadRaw_Error;
    if ((TWSR & 0xF8) != MR_SLA_ACK) goto ADC121C021_ReadRaw_Error;
    _delay_us(IC2_COM_DELAY_us);

    // Read MSB, ACK
    TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_ReadRaw_Error;
    if ((TWSR & 0xF8) != MR_DATA_ACK) goto ADC121C021_ReadRaw_Error;
    msb = TWDR;

```

```
140     // Read LSB, NACK
141     TWCR = (1 << TWINT) | (1 << TWEN);
142     if (!I2C_Wait_TWINT()) goto ADC121C021_ReadRaw_Error;
143     if ((TWSR & 0xF8) != MR_DATA_NAK) goto ADC121C021_ReadRaw_Error;
144     lsb = TWDR;
145 
146     // Stop
147     TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
148     _delay_us(IC2_COM_DELAY_us);
149 
150     *raw_out = (((uint16_t)msb << 8) | lsb) & 0x0FFF;
151     return 1;
152 
153 ADC121C021_ReadRaw_Error:
154     TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
155     _delay_us(IC2_COM_DELAY_us);
156     return 0;
157 }
158 
159 void ADC121C021_Set_Config(uint8_t address){
160         // Set configuration register
161         // Send start condition
162         TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
163         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Config_Sent_Stop; // Wait for start to
                be sent
164         if ((TWSR & 0xF8) != START) goto ADC121C021_Setup_Config_Sent_Stop; // Check correct
                status
165         _delay_us(IC2_COM_DELAY_us);
166 
167 
168         // send address
169         TWDR = (address << 1); // Write address
170         TWCR = (1<<TWINT)|(1<<TWEN); // Start transmission address
171         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Config_Sent_Stop; // Wait for start to
                be sent
172         if ((TWSR & 0xF8) != MT_SLA_ACK) goto ADC121C021_Setup_Config_Sent_Stop; // Check
                correct status
173         _delay_us(IC2_COM_DELAY_us);
174 
175         // send register
176         TWDR = ADC121C021_REG_CONFIG;
177         TWCR = (1 << TWINT) | (1 << TWEN);
178         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Config_Sent_Stop; // Wait for start to
                be sent
179         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Config_Sent_Stop;
180         _delay_us(IC2_COM_DELAY_us);
181 
182         // send data
183         TWDR = 0b11100010; // 0.4ksps auto conversion, Alert self clear,Enable alert ,Alert
                active low
184         TWCR = (1 << TWINT) | (1 << TWEN);
185         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Config_Sent_Stop; // Wait for start to
                be sent
186         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Config_Sent_Stop;
187         _delay_us(IC2_COM_DELAY_us);
188 
189         // send stop
190         ADC121C021_Setup_Config_Sent_Stop:
191         TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
192         _delay_us(IC2_COM_DELAY_us);
193 }
194 
195 void ADC121C021_Set_Vlow(uint8_t address){
196 
197         uint8_t msb;
198         uint8_t lsb;
199         uint16_t Vlow;
200         // Calculate Vlow threshold
201         Vlow = NTC_TempToRaw(TEMP_MAX_Heat_NTC);
202         Vlow &= 0x0FFF;
203 
```

```
204         lsb = (uint8_t) (Vlow & 0xFF);
205         msb = (uint8_t) (Vlow >> 8);
206 
207 
208         // Set V_low
209         // Send start condition
210         TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
211         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vlow_Sent_Stop; // Wait for start to be
                sent
212         if ((TWSR & 0xF8) != START) goto ADC121C021_Setup_Vlow_Sent_Stop; //Check correct
                status
213         _delay_us(IC2_COM_DELAY_us);
214 
215         //send address
216         TWDR = (address << 1); //Write address
217         TWCR = (1 << TWINT) | (1 << TWEN); //Start transmission address
218         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vlow_Sent_Stop; // Wait for start to be
                sent
219         if ((TWSR & 0xF8) != MT_SLA_ACK) goto ADC121C021_Setup_Vlow_Sent_Stop; //Check
                correct status
220         _delay_us(IC2_COM_DELAY_us);
221 
222         //send register
223         TWDR = ADC121C021_REG_V_LOW;
224         TWCR = (1 << TWINT) | (1 << TWEN);
225         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vlow_Sent_Stop; // Wait for start to be
                sent
226         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Vlow_Sent_Stop;
227         _delay_us(IC2_COM_DELAY_us);
228 
229         // send high byte
230         TWDR = msb;
231         TWCR = (1 << TWINT) | (1 << TWEN);
232         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vlow_Sent_Stop; // Wait for start to be
                sent
233         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Vlow_Sent_Stop;
234         _delay_us(IC2_COM_DELAY_us);
235 
236         // send low byte
237         TWDR = lsb;
238         TWCR = (1 << TWINT) | (1 << TWEN);
239         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vlow_Sent_Stop; // Wait for start to be
                sent
240         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Vlow_Sent_Stop;
241         _delay_us(IC2_COM_DELAY_us);
242 
243         // send stop
244         ADC121C021_Setup_Vlow_Sent_Stop:
245         TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
246         _delay_us(IC2_COM_DELAY_us);
247 }
248 
249 //ADC will deasert the alert pin if the voltage is X amount higher than Vlow or (X higher
        than Vhigh)
250 //Where X is the value stored at the hysteresis register
251 void ADC121C021_Set_Vhyst(uint8_t address){
252         // Calculate V Hysteresis
253         // Hysteresis is calculated based on 5 degrees below the upper temperature limit
254         float temp_hyst; // Temperature where the alert should be turned off
255         uint8_t msb;
256         uint8_t lsb;
257         uint16_t Vhyst;
258 
259         temp_hyst = TEMP_MAX_Heat_NTC - 5.0; // Take 5degrees below max temperature
260         Vhyst = NTC_TempToRaw(temp_hyst) - NTC_TempToRaw(TEMP_MAX_Heat_NTC);
261         Vhyst &= 0x0FFF;
262 
263         lsb = (uint8_t) (Vhyst & 0xFF);
264         msb = (uint8_t) (Vhyst >> 8);
265 
266 
```

```
267         //Set V_hyst
268         //Send start condition
269         TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
270         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vhyst_Sent_Stop; // Wait for start to be
                sent
271         if ((TWSR & 0xF8) != START) goto ADC121C021_Setup_Vhyst_Sent_Stop; //Check correct
                status
272         _delay_us(IC2_COM_DELAY_us);
273
274         //send address
275         TWDR = (address << 1) ; //Write address
276         TWCR = (1 << TWINT) | (1 << TWEN); //Start transmission address
277         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vhyst_Sent_Stop; // Wait for start to be
                sent
278         if ((TWSR & 0xF8) != MT_SLA_ACK) goto ADC121C021_Setup_Vhyst_Sent_Stop; //Check
                correct status
279         _delay_us(IC2_COM_DELAY_us);
280
281         //send register
282         TWDR = ADC121C021_REG_V_HYST;
283         TWCR = (1 << TWINT) | (1 << TWEN);
284         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vhyst_Sent_Stop; // Wait for start to be
                sent
285         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Vhyst_Sent_Stop;
286         _delay_us(IC2_COM_DELAY_us);
287
288         // send high byte
289         TWDR = msb;
290         TWCR = (1 << TWINT) | (1 << TWEN);
291         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vhyst_Sent_Stop; // Wait for start to be
                sent
292         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Vhyst_Sent_Stop;
293         _delay_us(IC2_COM_DELAY_us);
294
295         // send low byte
296         TWDR = lsb;
297         TWCR = (1 << TWINT) | (1 << TWEN);
298         if (!I2C_Wait_TWINT()) goto ADC121C021_Setup_Vhyst_Sent_Stop; // Wait for start to be
                sent
299         if ((TWSR & 0xF8) != MT_DATA_ACK) goto ADC121C021_Setup_Vhyst_Sent_Stop;
300         _delay_us(IC2_COM_DELAY_us);
301
302         // send stop
303         ADC121C021_Setup_Vhyst_Sent_Stop:
304         TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
305         _delay_us(IC2_COM_DELAY_us);
306 }
307
308 float NTC_RawToTemp(uint16_t raw){
309     float ratio;
310     float r_ntc;
311     float inv_t;
312     float temp_k;
313     float temp_c;
314
315     if (raw == 0 || raw >= 4095) {
316         return 999.9;   //Invalid return 999.9
317     }
318
319     ratio = (float)raw / 4095.0;
320
321     r_ntc = R_FIXED * ratio / (1.0 - ratio);
322
323     inv_t = 1.0 / 298.15 + log(r_ntc / R_NTC_T0) / NTCALUG0_BETA_K;
324     temp_k = 1.0 / inv_t;
325     temp_c = temp_k - 273.15;
326
327     return temp_c;
328 }
329
330
```

```
uint16_t NTC_TempToRaw(float temp_c){
        float temp_k;
    float r_ntc;
        float ratio;
        float raw;

        temp_k = temp_c + 273.15;

        r_ntc =  R_NTC_T0 * exp(NTCALUG0_BETA_K * (1/temp_k - 1.0/ 298.15));

        ratio = r_ntc / (r_ntc + R_FIXED);

        raw = 4095 * ratio;

        //Clamp result
        if (raw < 0.0) {
        raw = 0.0;
    }
    if (raw > 4095.0) {
        raw = 4095.0;
    }
        return (uint16_t) raw;
}


//Helper to handle I2C communication and give a timeout if it takes too long
uint8_t I2C_Wait_TWINT(){
    uint16_t timeout = I2C_TIMEOUT_COUNT;

    while (!(TWCR & (1 << TWINT))) {
        if (--timeout == 0) {
            return 0;
        }
    }

    return 1;
}

uint8_t ADC121C021_Is_Present(uint8_t address){
    // Send start
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_IsPresent_Error;
    if ((TWSR & 0xF8) != START) goto ADC121C021_IsPresent_Error;

    // Send address write
    TWDR = (address << 1);
    TWCR = (1 << TWINT) | (1 << TWEN);
    if (!I2C_Wait_TWINT()) goto ADC121C021_IsPresent_Error;

    if ((TWSR & 0xF8) != MT_SLA_ACK) {
        goto ADC121C021_IsPresent_Error;
    }

    // Stop
    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
    _delay_us(IC2_COM_DELAY_us);

    return 1;

        ADC121C021_IsPresent_Error:
    TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO);
    _delay_us(IC2_COM_DELAY_us);

    return 0;
}
```

Code Listing C.4: Header file for the power dissipation device

```
/*
        ----------------------------------------
        MPPT PCB MCU source
        ----------------------------------------
        Made by:        Justen van Koppen
        Version:        1.00
        Date:           20.05.2026
        ----------------------------------------
        ATMega 1284 Micro controller support
        ----------------------------------------
        Temperature monitoring Header FILE
        ========================================
*/

#ifndef MPPT_PCB_MCU__TEMP_H_
#define MPPT_PCB_MCU__TEMP_H_

//=============================================================================================
// Definitions and constants


#define I2C_TIMEOUT_COUNT 10000


//Registers
#define ADC121C021_REG_CONV_RESULT 0x00
#define ADC121C021_REG_CONFIG 0x02
#define ADC121C021_REG_V_LOW 0x03
#define ADC121C021_REG_V_HIGH 0x04
#define ADC121C021_REG_V_HYST 0x05

//Vishay NTCALUG01A103J
#define R_FIXED 10000.0   // fixed divider resistor
#define R_NTC_T0  10000.0   // NTC resistance at 25 C
#define NTCALUG0_BETA_K  3984.0    // beta value from NTC datasheet NTCALUG01A103J

//=============================================================================================
// VARIABLES and STRUCTURES

extern const uint8_t ADDRESSES_ADC121C021[8];

extern volatile float AI_HEAT_NTC_Temp[8];
extern volatile uint8_t AI_HEAT_ADC_Present_Mask;
extern volatile uint8_t AI_HEAT_ADC_Fault_Mask;

//=============================================================================================
// EXTERN EEPROM VARIABLES


//=============================================================================================
// FUNCTION Prototypes

void Temp_monitoring_Setup(void);
void Read_Temp_ADC121C021(uint8_t TEMP_i_Device);

void ADC121C021_Setup(uint8_t address);
uint8_t ADC121C021_ReadRaw(uint8_t i, uint16_t *raw_out);
void ADC121C021_Set_Config(uint8_t address);
void ADC121C021_Set_Vlow(uint8_t address);
void ADC121C021_Set_Vhyst(uint8_t address);
float NTC_RawToTemp(uint16_t raw);
uint16_t NTC_TempToRaw(float temp_c);
uint8_t ADC121C021_Is_Present(uint8_t address);
uint8_t I2C_Wait_TWINT(void);

//end
#endif /* MPPT_PCB_MCU__TEMP_H_ */
```

D Software Source Code

D.1. Database

D.1.1. Update loop

Code Listing D.1: Function that runs every 10 seconds and adds data

```
def update_loop():
    """ This loop adds newly collected data to the database every 10s."""
    conn, cur, mysql_conn, mysql_cur= init()
    while(1):
        config, data_path_base = loadconfig()
        date = str(datetime.date.today())
        try:
            add_data(date, conn, cur ,mysql_conn , mysql_cur, config, data_path_base)
            count_entries("pv_point", conn, cur)
            count_entries('pv_curve', conn, cur)
        except Exception as e:
            print(f"Data could not be added, maybe the file is not yet created or there is an
                 error: {e}")
            logger.error(f"Data could not be added, maybe the file is not yet created or
                there is an error: {e}")
            conn.rollback()

        try:
            add_weather_data(download_weather_last24hours(1, mysql_conn, mysql_cur), conn,
                cur)
            logger.debug("Weather data succesfully added")
        except Exception as e:
            print(f"Weather data could not be added. Error: {e}")
            logger.error(f"Weather data could not be added. Error {e}")
            conn.rollback()

        try:
            update_weather_id(date, conn, cur)
            logger.debug(f"Succesfully updated the weather_id")
        except Exception as e:
            logger.error(f"Weather_id could not be updated. Error {e}")
        time.sleep(10) # Wait for an 10s and check again.
    db_close(conn)
```

D.1.2. Daily loop

Code Listing D.2: Function that runs everyday, adds data retroactively, and detects errors

```
def daily_loop():
    """ This loop uploads past data at midnight so that data that was missed still gets
        uploaded.
        It also checks for errors in the system.
    """
    conn, cur, mysql_conn, mysql_cur= init()
    while(1):
        config, data_path_base = loadconfig()
        if datetime.datetime.now().hour == 0: #At midnight
            try:
                past_data_upload(conn, cur, mysql_conn, mysql_cur, config, data_path_base)
            except Exception as e:
                print("Data could not be added, maybe the file is not yet created or there is
                     an error")
                logging.error(f"Past data could not be added. Error: {e}")
                conn.rollback()
```

```
15
16              try:
17                  error_detect(conn, cur, config)
18              except Exception as e:
19                  print(f"Error detection failed with error: {e}")
20                  logging.error(f"Error detection failed with error: {e}")
21                  conn.rollback()
22          time.sleep(60*60*1) # Wait for an hour and check again.
23      db_close(conn)
```

Code Listing D.3: Function that adds the past OPET and weather data

```
1  def past_data_upload(conn, cur, mysql_conn, mysql_cur, config, data_path_base):
2      """ This function adds all the data to the database from a lookback number of days back.
3
4      Args:
5          conn (_type_): The connection to the PostgreSQL database.
6          cur (_type_): The cursor for the PostgreSQL database.
7          mysql_conn (_type_): The connection to the MySQL database.
8          mysql_cur (_type_): The cursor for the MySQL database.
9          config (dict): The measurement config.
10         data_path_base (_type_): The base location of the files.
11     """
12
13     start_date = datetime.date.today()-datetime.timedelta(days=config.get('lookback', 7))
14     try:
15         add_weather_data(weather_all(start_date, mysql_conn, mysql_cur), conn, cur)
16         logger.debug("Past weather data succesfully uploaded")
17     except Exception as e:
18         print('Could not add the weather data')
19         logger.error(f"Weather data could not be added. Error: {e}")
20         conn.rollback()
21
22     for date in (start_date + datetime.timedelta(days=n) for n in range((datetime.date.today
           () - start_date + datetime.timedelta(days=1)).days)):
23         try:
24             add_data(str(date), conn, cur, mysql_conn, mysql_cur , config, data_path_base)
25         except Exception as e:
26             print("Data could not be added, maybe the file is not yet created or there is an
                   error")
27             logger.error(f"Date could not be added on {date}. Error: {e}")
28             conn.rollback()
29         try:
30             update_weather_id(str(date), conn, cur)
31             logger.debug(f"weather_id succesfully updated")
32         except Exception as e:
33             logger.error(f"Weather_id not succesfully updated on {date}. Error: {e}")
34             print('weather_id not succesfully updated')
35             conn.rollback()
```

D.1.3. Add Data

Code Listing D.4: Function that adds the data to the database

```
1  def add_data(date, conn, cur, mysql_conn, mysql_cur, config, data_path_base):
2      """ Adds the data on a specifc date to the database
3
4      Args:
5          date (_string_): The date for which the data has to be added, example: "2026-05-20"
6          conn (_type_): The connection to the PostgreSQL database
7          cur (_type_): The cursor for the PostgreSQL database
8          mysql_conn (_type_): The connection to the MySQL database
9          mysql_cur (_type_): The cursor for the MySQL database
10         config (_dict_): The measurement config
11         data_path_base (_type_): The base location of the files.
12     """
13     print(date)
14
15     # Every time new measurement data gets added first for a new module has to be checked.
16     add_module_data(config, conn, cur)
17
```

```
    data_path = data_path_base / date / config['data_destination']

    #point data add
    data_file_path = (
        data_path / (
        'opet_results_'
        + 'point'
        + '_' + date
        + '.csv'
        )
    )
    df = pd.read_csv(data_file_path, delimiter=",")

    # Add weather_id column with None values, this is done because the weather data is not always available,
    # so the weather_id will be added later when the weather data is available, .
    # By adding the column with None values, the data can still be added to the database without having to worry about missing weather data.
    df['weather_id'] = None
    df['date_time']=pd.to_datetime(df['date_time'])
    df['scheduled_time'] = pd.to_datetime(df['scheduled_time'])

    # Insert the point data into the database
    data = df.to_numpy()
    point_insert = (
        "INSERT INTO pv_point (date_time, scheduled_time, module_name, mounted_on, v, i, status_integer, temperature_cell, axis_azimuth, axis_tilt, weather_id) "
        "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s) "
        "ON CONFLICT (date_time, module_name) DO NOTHING"
    )
    for d in data:
        cur.execute(point_insert, d)
    conn.commit()
    print('point data added')

    # Curve data add
    #Open document
    data_file_path = (
        data_path / (
        'opet_results_'
        + 'curve'
        + '_' + date
        + '.csv'
        )
    )
    df = pd.read_csv(data_file_path, delimiter=",")

    #From string to vector
    df["v"] = df["v"].apply(ast.literal_eval)
    df['i'] = df['i'].apply(ast.literal_eval)

    #Assigning weather_id None
    df['weather_id'] = None
    df['date_time']=pd.to_datetime(df['date_time'])
    df['scheduled_time'] = pd.to_datetime(df['scheduled_time'])

    # Insert the curve data into the database
    data = df.to_numpy()
    curve_insert = (
        "INSERT INTO pv_curve (date_time, scheduled_time, measurement_duration, module_name, mounted_on, v, i, iv_status_integer, temperature_cell, axis_azimuth, axis_tilt, weather_id) "
        "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s) "
        "ON CONFLICT (date_time, module_name) DO NOTHING"
    )
    for d in data:
        cur.execute(curve_insert, d)
    conn.commit()
    print('curve data added')
```

Code Listing D.5: Function that adds the module data to the database

```
def add_module_data(config, conn, cur):
    """ Adds the module data to the modules table

    Args:
        config (dict): The measurement config
        conn (_type_): The connection to the PostgreSQL database
        cur (_type_): The cursor for the PostgreSQL database
    """
    for module in config['modules']:
        module_insert = (
            "INSERT INTO modules (module_name, tracer, username, user_email, area, technology
                , manufacturer) "
            "VALUES (%s, %s, %s, %s, %s, %s, %s) "
            "ON CONFLICT (module_name) DO NOTHING" # If there is already a module with the
                same name, the data will not be added.
        )
        query = "SELECT * FROM modules WHERE module_name = %s"
        cur.execute(query, (module['module_name'],))
        module_db = cur.fetchone()
        #print(module_db)
        conn.commit()
        try:
            if (module['module_name'] == module_db[0] and module.get('area') == module_db[4]
                and
            module.get('technology') == module_db[5] and module.get('manufacturer') ==
                module_db[6]):
                cur.execute("UPDATE modules SET tracer = %s, username= %s, user_email = %s",
                    (module.get('tracer'), module.get('username'), module.get('user_email')))
                conn.commit()
                logger.debug(f"Updated the tracer, username and user_email")
            elif module['module_name'] == module_db[0]:
                #print('Module_name already exists')
                logger.debug('Duplicate module_name')
        except:
            logger.debug("Data could not be updated")
            logger.debug(f"Added a new module")
            cur.execute(module_insert, (module['module_name'], module.get('tracer'), module.
                get('username'), module.get('user_email'), module.get('area'), module.get('
                technology'), module.get('manufacturer')))
            conn.commit()
```

Code Listing D.6: Function that adds the weather data to the database

```
def add_weather_data(weather_data, conn, cur):
    """ Add the weather data collected to the weather table

    Args:
        weather_data (list): The weather data is to be added to the database
        conn (_type_): The connection to the PostgreSQL database
        cur (_type_): The cursor for the PostgreSQL database
    """
    weather_insert = (
        "INSERT INTO weather (weather_id, weather_time, temperature_air, relative_humidity,
            dew_point, relative_pressure, wind_speed, wind_speed_std, wind_direction,
            wind_direction_std, irradiance) "
        "VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s) "
        "ON CONFLICT (weather_id) DO NOTHING"
    )
    df = pd.DataFrame(weather_data, columns=['weather_id', 'weather_time', 'temperature_air',
         'relative_humidity', 'dew_point', 'relative_pressure', 'wind_speed', 'wind_speed_std
        ', 'wind_direction', 'wind_direction_std', 'irradiance', 'IrrDirect', 'IrrDiffused',
        'ETotal', 'EDirect', 'EDiffused'])
    df = df.drop(['IrrDirect', 'IrrDiffused', 'ETotal', 'EDirect', 'EDiffused'], axis=1) #
        drop the columns that are not needed.
    df['weather_time'] = pd.to_datetime(df['weather_time']).dt.tz_localize('Europe/Amsterdam'
        , ambiguous='NaT', nonexistent='NaT')
    df['weather_time'] = df['weather_time'].astype(object).where(df['weather_time'].notna(),
        None)  # When there is a glitch with the time. This should make it None.
    result = df.to_numpy()

```

```
    for d in result:
        cur.execute(weather_insert, d) # the first column is the weather_id which is
            automatically generated by the database and is not needed to be added.
    conn.commit()
```

D.1.4. Weather ID synchronisation

Code Listing D.7: Function that updates the weather ID for the OPET measurements

```
def update_weather_id(date, conn, cur):
    """Assigns a weather_id to a opet measurement. It first checks for weather_id's that have
         not been filled in yet.
        Then it checks whether the measurement is 5 minutes or longer ago. Then it will
            assign the best fitting weather_id.

    Args:
        date (string): String that contains a date. Example: '2026-05-20'.
        conn (_type_): The connection to the PostgreSQL database.
        cur (_type_): The cursor for the PostgreSQL database.
        mysql_conn (_type_): The connection to the MySQL database.
        mysql_cur (_type_): The cursor for the MySQL datbase.
    """
    #Syncing for the pv_point measurements
    query1_point = "SELECT date_time FROM pv_point WHERE date_time>%s AND date_time<%s AND
        weather_id IS NULL"
    naive_date = (datetime.datetime.strptime(date, "%Y-%m-%d"))
    aware_date= naive_date.replace(tzinfo=zoneinfo.ZoneInfo('Europe/Amsterdam'))
    aware_date2 = aware_date+datetime.timedelta(days=1)
    cur.execute(query1_point, (aware_date, aware_date2))
    datetimes = cur.fetchall()
    conn.commit()

    for time in datetimes:
        time = time[0]
        now = datetime.datetime.now(tz=zoneinfo.ZoneInfo('Europe/Amsterdam'))
        if now-time>datetime.timedelta(minutes = 5):
            weatherid = weather_sync(time, conn, cur)
            query2_point = "UPDATE pv_point SET weather_id = %s WHERE date_time=%s" #Use the
                local weatherdb for speed.
            cur.execute(query2_point, (weatherid, time))
            conn.commit()

    query1_curve = "SELECT date_time FROM pv_curve WHERE date_time>%s AND date_time<%s AND
        weather_id IS NULL"
    naive_date = (datetime.datetime.strptime(date, "%Y-%m-%d"))
    aware_date= naive_date.replace(tzinfo=zoneinfo.ZoneInfo('Europe/Amsterdam'))
    aware_date2 = aware_date+datetime.timedelta(days=1)
    cur.execute(query1_curve, (aware_date, aware_date2))
    datetimes = cur.fetchall()
    conn.commit()

    for time in datetimes:
        time = time[0]
        now = datetime.datetime.now(tz=zoneinfo.ZoneInfo('Europe/Amsterdam'))
        if now-time>datetime.timedelta(minutes = 5):
            weatherid = weather_sync(time, conn, cur)
            query2_curve = "UPDATE pv_curve SET weather_id = %s WHERE date_time=%s" #Use the
                local weatherdb for speed.
            cur.execute(query2_curve, (weatherid, time))
            conn.commit()
```

Code Listing D.8: Function that collects most closest in time weather ID

```
def weather_sync(opet_date_time, conn, cursor):
    """ This function returns a weather id that is within a 5 minute range of a OPET data
        measurement.
        If no weather measurement is within that timespan a 0 is returned.

    Args:
        opet_date_time (_datetime_): The date and time of the OPET measurement for which the
            weather_id has to be found. This should be a timezone aware datetime in the
```

```
            timezone of Europe/Amsterdam.
        conn (_type_): The connection to the PostgreSQL database
        cursor (_type_): The cursor for the PostgreSQL database

    Returns:
        _int_: The weather_id that is within a 5 minute range. 0 if no weather measurement is
             within that timespan.
    """

    cursor.execute("SELECT weather_id, weather_time FROM weather ORDER BY ABS(EXTRACT(EPOCH
        FROM (weather_time - %s))) ASC limit 1", (opet_date_time,))
    data = cursor.fetchone()
    data = np.array(data)
    #print(data)
    data[1]=data[1].replace(tzinfo=zoneinfo.ZoneInfo('Europe/Amsterdam'))
    if abs(opet_date_time-data[1])  < datetime.timedelta(minutes = 5):
        weatherid=data[0]
    else:
        weatherid= 0

    return weatherid
```

D.1.5. Error Detection

Code Listing D.9: Function that detects the error and send an email

```
def error_detect(conn, cur, config):
    """ This function detects error in the system such as: problems with the entire system
        and issues per module.
        When it has detected a problem it sends an email.

    Args:
        conn (_type_): The connection to the PostgreSQL database
        cur (_type_): The cursor for the PostgreSQL database
        config (dict): The measurement config to load the setup.
    """
    point_measurements = True
    curve_measurements = True
    # Check if data is being collected, if not send an email to the admin.
    cur.execute("SELECT date_time FROM pv_point ORDER BY date_time DESC LIMIT 1")
    last_entry_point = cur.fetchone()[0]
    cur.execute("SELECT date_time FROM pv_curve ORDER BY date_time DESC LIMIT 1")
    last_entry_curve = cur.fetchone()[0]

    # Check if curve and/or point measurements are being received
    if ((last_entry_point < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) -
        datetime.timedelta(days=1))) and
        (last_entry_curve < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) -
            datetime.timedelta(days=1)))):
        send_mail('The PV monitoring system has not received data from both point and curve
            measurements in the past 24 hours \n'
                +'Most recent data from point measurements: ' + str(last_entry_point)
                +'\nMost recent data from curve measurements: ' + str(last_entry_curve),
                    config
                )
        point_measurements = False
        curve_measurements = False
    elif last_entry_point < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) -
        datetime.timedelta(days =1)):
        send_mail('The PV monitoring system database has not received data from point
            measurements in the past 24 hours \n'
                +'Most recent data from point measurements: '
                + str(last_entry_point), config
                )
        point_measurements = False
    elif last_entry_curve < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) -
        datetime.timedelta(days=1)):
        send_mail('The PV monitoring system database has not received data from curve
            measurements in the past 24 hours.\n'
                +'Most recent data from curve measurements: '
                + str(last_entry_curve), config
```

```
                )
        curve_measurements = False

    # If data is collected check whether the OPETs are suffering from errors, ie status_integer is not 1.
    # If there are errors, send an email to the user of the OPET and the admin.
    # Send the number of errors in the last 24 hours and the total number of measurements in the last 24 hours.
    for module in config['modules']:
        cur.execute("SELECT date_time, status_integer FROM pv_point WHERE date_time > %s AND module_name = %s ORDER BY date_time DESC",
                (str(datetime.datetime.now() - datetime.timedelta(days=1)),) + (module['module_name'],)
                )
        last_24h = cur.fetchall()
        errorcount = 0
        for i in range(len(last_24h)):
            if last_24h[i][1] != 1:
                errorcount += 1
        if errorcount > 0: # if there is at least one error in the last 24 hours, send an email
            if module.get('disabled', False) == False:
                send_mail(module['module_name']+' Has had an error in the past 24 hours, please check the system. \n'
                     +str(errorcount)+' of '+str(len(last_24h))
                     +' measurements have had an error in the past 24 hours', config, module.get('user_email', '')
                     )

        # Check whether the gets collected from the individual modules
        if module.get('disabled', False) == False:
            cur.execute("SELECT date_time FROM pv_point WHERE module_name = %s ORDER BY date_time DESC LIMIT 1", (module['module_name'],))
            last_entry_point = cur.fetchone()[0]
            cur.execute("SELECT date_time FROM pv_curve WHERE module_name = %s ORDER BY date_time DESC LIMIT 1", (module['module_name'],))
            last_entry_curve = cur.fetchone()[0]
            if (last_entry_point < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) - datetime.timedelta(days=1))
                and last_entry_curve < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) - datetime.timedelta(days=1))
                and point_measurements == True and curve_measurements == True):
                send_mail(module['module_name']+' on tracer: '+module.get('tracer', 'unknown tracer')
                     + ' has not received data from both point and curve measurements in the past 24 hours \n'
                     + 'Most recent data from point measurements: '
                     + str(last_entry_point)
                     + '\nMost recent data from curve measurements: '
                     + str(last_entry_curve),
                     config, module.get('user_email', '')
                     )
            elif (last_entry_point < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) - datetime.timedelta(days=1)) and
                  point_measurements == True):
                send_mail(module['module_name']+' on tracer: '+module.get('tracer', 'unknown tracer')
                     + ' has not received data from point measurements in the past 24 hours \nMost recent data from point measurements: '
                     + str(last_entry_point),
                     config, module.get('user_email', '')
                     )
            elif (last_entry_curve < (datetime.datetime.now(zoneinfo.ZoneInfo("Europe/Amsterdam")) - datetime.timedelta(days=1)) and
                  curve_measurements == True):
                send_mail(module['module_name']+' on tracer: '+module.get('tracer', 'unknown tracer')
                     + ' has not received data from curve measurements in the past 24 hours\nMost recent data from curve measurements: '
                     + str(last_entry_curve),
                     config, module.get('user_email', '')
```

```
                        )
```

Code Listing D.10: Function that sends emails

```
def send_mail(error, config, receiver_email = ''):
    """ Send an email with a specific message to a specific person and to admins.

    Args:
        error (string): The message that you want to send
        config (dict): The measurement config to load the setup.
        receiver_email (str, optional): additional emailaddress to send an email to. Defaults
             to ''.
    """

    smpt_server = "smtp.gmail.com"
    port = 587
    sender_email = 'johndoe@gmail.com'
    password = 'password'
    admin = config['admins_email']
    if receiver_email == '':
        logging.debug(f'No useremail to send to')
        users= receiver_email + ',' + admin
    elif '@' in receiver_email and '.' in receiver_email:
        users = receiver_email + ',' + admin
        logging.debug(f"Email address is correct")
    else:
        receiver_email = ''
        users = receiver_email + ',' + admin
        logging.error(f"Emailaddress is not an emailaddress.")
    message = MIMEMultipart()
    message['From'] = sender_email
    message['To'] = users
    message['Subject'] = 'There is a problem with the PV monitoring system'
    body = 'Error: ' + str(error)
    message.attach(MIMEText(body, 'plain'))


    try:
        server = smtplib.SMTP(smpt_server, port)
        server.starttls()
        server.login(sender_email, password)
        server.send_message(message)
        print("Email sent successfully")
        server.quit()
        logging.debug(f"Email has been send")

    except Exception as e:
        print(f"Error sending email: {e}")
        logging.error(f"Error sending email : {e}")
```

D.1.6. Data Extraction

Code Listing D.11: Function that extracts a CSV from the database

```
def download_table(file, type, datetime1, datetime2, module_name, conn, cur):
    """This function gives the ability to download the data from the database with filters
        over time and over modules.

    Args:
        file (csv): 'file_name.csv'
        type (string): Measurement type: 'pv_point' or 'pv_curve'.
        datetime1 (string): start datetime in string, example: "2024-12-20 16:00:50-07:00".
        datetime2 (string): end datetime in string, example: "2026-12-20 16:00:50-07:00".
        module_name (list): List of the modules selected.
        conn (_type_): Connection to the PostgreSQL database.
        cur (_type_): cursor for the PostgreSQL database.
    """

    dt1 = datetime.datetime.fromisoformat(datetime1)
    dt2 = datetime.datetime.fromisoformat(datetime2)
```

```
    query = sql.SQL("COPY (SELECT * FROM " +type+ " LEFT JOIN weather ON "+type+".weather_id
        = weather.weather_id WHERE scheduled_time > %s AND scheduled_time < %s ORDER BY
        date_time DESC) TO STDOUT WITH DELIMITER ',' CSV HEADER ")
    with open(file, 'w') as f:
        formatted_query = cur.mogrify(query, [dt1,dt2]).decode('utf-8')
        cur.copy_expert(formatted_query, f)
    df = pd.read_csv(file)
    df['scheduled_time'] = pd.to_datetime(df['scheduled_time'])
    df.drop('weather_id.1', axis=1, inplace=True) # Drop the double weahter_id column
    result = df.loc[(df['module_name'].isin(module_name))]
    print(result)
    result.to_csv(file, index=False)
```

Code Listing D.12: Script that plots the measurements

```
"""
DISCLAIMER: GENERATED BY AI
Live solar panel monitoring with three stacked plots:
    1. Voltage
    2. Current
    3. Cell temperature

This script combines:
- the live database polling / matplotlib animation from live_data.py
- the three-subplot layout from plot_csv.py

Change MODULE_NAME below to choose which solar module is plotted.
"""

from __future__ import annotations

import datetime as dt
import zoneinfo
from collections import deque
from typing import Optional, Sequence

import matplotlib
import matplotlib.dates as mdates
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

from tud_pv_monitoring.database import init, db_close


# ---------------------------------------------------------------------------
# User settings
# ---------------------------------------------------------------------------
MODULE_NAME = "My_solar_panel_1"
TIMEZONE = "Europe/Amsterdam"
POLL_INTERVAL_MS = 1_000          # Query the database every 3 seconds
WINDOW_DURATION = dt.timedelta(minutes=1)
MAX_POINTS = 1_000                # Prevent unlimited memory growth

# Database column names. These match the CSV columns used in plot_csv.py.
# If your database uses different names, change them here.
TIME_COL = "date_time"
VOLTAGE_COL = "v"
CURRENT_COL = "i"
TEMP_COL = "temperature_cell"
TABLE_NAME = "pv_point"
MODULE_COL = "module_name"


# ---------------------------------------------------------------------------
# Database helpers
# ---------------------------------------------------------------------------
def last_measurement(cur, module_name: str) -> Optional[tuple]:
    """
    Return the latest measurement for one module as:
        (date_time, voltage, current, temperature_cell)

```

```
    The old live_data.py selected every column with SELECT * and then used
    numeric indexes. This version selects only the needed columns by name,
    which is safer and easier to maintain.
    """
    query = f"""
        SELECT {TIME_COL}, {VOLTAGE_COL}, {CURRENT_COL}, {TEMP_COL}
        FROM {TABLE_NAME}
        WHERE {MODULE_COL} = %s
        ORDER BY {TIME_COL} DESC
        LIMIT 1
    """
    cur.execute(query, (module_name,))
    return cur.fetchone()


def to_local_datetime(value, timezone_name: str) -> dt.datetime:
    """Ensure timestamps are timezone-aware for correct matplotlib formatting."""
    timezone = zoneinfo.ZoneInfo(timezone_name)

    if isinstance(value, str):
        value = dt.datetime.fromisoformat(value)

    if value.tzinfo is None:
        return value.replace(tzinfo=timezone)

    return value.astimezone(timezone)


# ---------------------------------------------------------------------------
# Plot setup
# ---------------------------------------------------------------------------
matplotlib.set_loglevel("warning")

conn = None
mysql_conn = None
animation = None
last_seen_time = None

# Deques automatically discard old points after MAX_POINTS.
times = deque(maxlen=MAX_POINTS)
voltages = deque(maxlen=MAX_POINTS)
currents = deque(maxlen=MAX_POINTS)
temperatures = deque(maxlen=MAX_POINTS)

fig, axes = plt.subplots(
    3,
    1,
    figsize=(12, 8),
    sharex=True,
    constrained_layout=True,
)

voltage_line, = axes[0].plot([], [], marker="o", linewidth=1, markersize=3, label="Voltage")
current_line, = axes[1].plot([], [], marker="o", linewidth=1, markersize=3, label="Current")
temperature_line, = axes[2].plot([], [], marker="o", linewidth=1, markersize=3, label="Cell
    temperature")

plot_config = [
    (axes[0], voltage_line, "Voltage [V]"),
    (axes[1], current_line, "Current [A]"),
    (axes[2], temperature_line, "Cell temperature [°C]"),
]

for ax, line, ylabel in plot_config:
    ax.set_ylabel(ylabel)
    ax.grid(True, alpha=0.3)
    ax.legend(loc="upper left")

axes[-1].set_xlabel("Time")
fig.suptitle(f"Live solar panel measurements: {MODULE_NAME}")

```

```
# Format x-axis as local clock time.
timezone = zoneinfo.ZoneInfo(TIMEZONE)
locator = mdates.AutoDateLocator()
formatter = mdates.DateFormatter("%H:%M:%S", tz=timezone)
axes[-1].xaxis.set_major_locator(locator)
axes[-1].xaxis.set_major_formatter(formatter)
fig.autofmt_xdate()


def update(_frame) -> Sequence:
    """Fetch the newest database row and update all three live plots."""
    global last_seen_time

    measurement = last_measurement(cur, MODULE_NAME)

    if measurement is None:
        print(f"No measurements found for module: {MODULE_NAME}")
        return voltage_line, current_line, temperature_line

    measurement_time, voltage, current, temperature = measurement
    measurement_time = to_local_datetime(measurement_time, TIMEZONE)

    # Avoid plotting the same database row repeatedly between new measurements.
    if measurement_time == last_seen_time:
        return voltage_line, current_line, temperature_line

    last_seen_time = measurement_time
    print(measurement)

    times.append(measurement_time)
    voltages.append(voltage)
    currents.append(current)
    temperatures.append(temperature)

    voltage_line.set_data(times, voltages)
    current_line.set_data(times, currents)
    temperature_line.set_data(times, temperatures)

    for ax in axes:
        ax.relim()
        ax.autoscale_view(scalex=False, scaley=True)

    # Show a scrolling time window once at least one point exists.
    if times:
        axes[-1].set_xlim(times[-1] - WINDOW_DURATION, times[-1])

    return voltage_line, current_line, temperature_line


def on_close(_event) -> None:
    """Close database connections when the matplotlib window is closed."""
    if conn is not None and mysql_conn is not None:
        db_close(conn, mysql_conn)


if __name__ == "__main__":
    conn, cur, mysql_conn, mysql_cur = init()
    fig.canvas.mpl_connect("close_event", on_close)

    animation = FuncAnimation(
        fig,
        update,
        interval=POLL_INTERVAL_MS,
        cache_frame_data=False,
    )

    try:
        plt.show()
    finally:
        # Also close connections if the program exits without a close_event.
        if conn is not None and mysql_conn is not None:
```

```
198             db_close(conn, mysql_conn)
```

D.2. Supervisor

D.2.1. Main supervisor script

Code Listing D.13: Main supervisor script

```
1  import multiprocessing
2  import datetime
3  from tud_pv_monitoring.measurement_scheduling_tools import datetime_range, present,
       next_occurrence
4  import json
5  from tud_pv_monitoring.database import daily_loop, update_loop
6  from tud_pv_monitoring.measurements.opet_supervisor_tools import measurement_loop,
       writer_loop
7  import logging
8  import traceback
9  from zoneinfo import ZoneInfo
10 import sys
11 from pathlib import Path
12
13 # Constants
14 # Measurements more than this far into the future will be handled on a
15 # subsequent loop
16 MAXIMUM_WAIT = 0.1  # s
17 # Infinite loops with nothing to do will delay this much before checking again
18 MINIMUM_WAIT = 0.01  # s
19 # Measurements are scheduled only this far into the future
20 SCHEDULE_HORIZON = 32  # s
21 # The schedule is updated this often
22 SCHEDULE_INTERVAL = 30  # s
23 #Maximum time a job may exist
24 MAX_JOB_TIME = datetime.timedelta(minutes=5)
25
26 TZ_LOCAL = ZoneInfo("Europe/Amsterdam")
27
28 # Load from .json files
29 with open('opet-supervisor-config.json') as f:
30     opet_supervisor_config = json.load(f)
31
32 CONFIG_PATH = Path(opet_supervisor_config['config_path'])
33 DATA_PATH_BASE = Path(opet_supervisor_config['data_path_base'])
34 LOG_PATH_BASE = Path(opet_supervisor_config['log_path_base'])
35
36 with open(CONFIG_PATH / 'opet_info.json') as f:
37     load_info = json.load(f)
38
39 with open(CONFIG_PATH / 'opet_bus_info.json') as f:
40     bus_info = json.load(f)
41
42 with open(CONFIG_PATH / 'measurement_config.json') as f:
43     config = json.load(f)
44
45
46 # Setup logging
47 LOG_PATH_BASE.mkdir(parents=True, exist_ok=True)
48 logger = logging.getLogger(__name__)
49 logging.basicConfig(
50     filename = (
51         LOG_PATH_BASE
52         / f'opet-supervisor-{datetime.date.today().isoformat()}.log'
53     ),
54     encoding = 'utf-8',
55     level=logging.DEBUG,
56     format='%(asctime)s %(levelname)s %(message)s'
57 )
58
59 logger.debug('opet-supervisor started')
60
61 # Set up a handler to log uncaught exceptions
```

```
def exception_handler(exctype, value, tb):
    logger.exception(''.join(traceback.format_exception(exctype, value, tb)))

sys.excepthook = exception_handler


if __name__ == '__main__':
    with multiprocessing.Manager() as manager:
        # Job definitions will be values in this shared dictionary.
        # Keys are unique job IDs so they can be removed from the dictionary
        # even if the dictionary gets updated while a job is underway. The
        # Manager doesn't handle updates to *nested* dictionaries for other
        # processes, so this code tends to pop things out of dictionaries,
        # work with them, then reinsert them if needed
        jobs = manager.dict({})
        results = manager.dict({})
        jobs_in_progress = manager.dict({})
        job_id = 0

        shutdown_event = manager.Event()
        # All possible buses. Each bus will get a process
        buses_all = {
            load_info_datum["bus"]
            for load_info_datum in load_info.values()
        }

        # Create a process for each bus
        processes = [
            multiprocessing.Process(
                target=measurement_loop,
                args=(
                    bus,
                    jobs,
                    jobs_in_progress,
                    results,
                    bus_info,
                    load_info,
                    MAXIMUM_WAIT,
                    MINIMUM_WAIT,
                    shutdown_event,
                    MAX_JOB_TIME
                ),
                name=f'measurement-bus-{bus}',
            )
            for bus
            in buses_all
        ]

        # Add one to do the logging
        processes.append(
            multiprocessing.Process(
                target=writer_loop,
                args=(
                    results,
                    DATA_PATH_BASE,
                    TZ_LOCAL,
                    MINIMUM_WAIT,
                    shutdown_event
                    ),
                name='writer',
            )
        )

        # Process for daily loop
        processes.append(
            multiprocessing.Process(
                target=daily_loop,
                name='daily_loop'
            )
        )

```

```
133         # Process for update loop database
134         processes.append(
135             multiprocessing.Process(
136                 target=update_loop,
137                 name='update_loop'
138             )
139         )
140 
141         for process in processes:
142             process.start()
143 
144         schedule_update_time = next_occurrence(
145             present(),
146             datetime.timedelta(seconds=SCHEDULE_INTERVAL)
147         )
148         schedule_never_updated = True
149         try:
150             while not shutdown_event.is_set():
151                 if schedule_never_updated:
152                     schedule_never_updated = False
153                 else:
154                     while present() < schedule_update_time and not shutdown_event.is_set():
155                         shutdown_event.wait(MINIMUM_WAIT)
156                     schedule_update_time += datetime.timedelta(seconds=SCHEDULE_INTERVAL)
157 
158                 # Check if processes are still active and restart inactive ones
159                 for i, process in enumerate(processes):
160                     if shutdown_event.is_set():
161                         break
162 
163                     if process.is_alive():
164                         continue
165 
166                     logger.error(f'Process {process.name} died: restarting')
167                     process.join(timeout=1)
168 
169                     old_name = process.name
170 
171 
172                     # Make new process based on name
173                     if old_name.startswith('measurement-bus-'):
174                         bus = old_name.removeprefix('measurement-bus-')
175 
176                         new_process = multiprocessing.Process(
177                             target=measurement_loop,
178                             args=(
179                                 bus,
180                                 jobs,
181                                 jobs_in_progress,
182                                 results,
183                                 bus_info,
184                                 load_info,
185                                 MAXIMUM_WAIT,
186                                 MINIMUM_WAIT,
187                                 shutdown_event,
188                                 MAX_JOB_TIME
189                             ),
190                             name=old_name,
191                         )
192 
193                     elif old_name == 'writer':
194                         new_process = multiprocessing.Process(
195                             target=writer_loop,
196                             args=(
197                                 results,
198                                 DATA_PATH_BASE,
199                                 TZ_LOCAL,
200                                 MINIMUM_WAIT,
201                                 shutdown_event
202                             ),
203                             name='writer',
```

```
                        )

                    elif old_name == 'daily_loop':
                        new_process = multiprocessing.Process(
                            target=daily_loop,
                            name='daily_loop'
                        )
                    elif old_name == 'update_loop':
                        new_process = multiprocessing.Process(
                            target=update_loop,
                            name='update_loop'
                        )
                    else:
                        logger.error(f'unknown child process name {old_name}; cannot restart'
                            )
                        continue

                    new_process.start()
                    processes[i] = new_process

                    logger.info(f'restarted {new_process.name}')

                # Reload the configuration
                try:
                    with open(CONFIG_PATH / 'measurement_config.json') as f:
                        new_config = json.load(f)
                except Exception:
                    logger.exception('Could not open the measurement config file')
                    new_config = config

                config = new_config

                #Log errror if measurement is not setup correctly
                for module in config['modules']:
                    tracer = module.get('tracer')
                    mounted_on = module.get('mounted_on')


                    if not tracer or not module.get('tracer').startswith('O'):
                        logger.error(f'{module.get("module_name")} does not have an expected
                            tracer assigned')
                        module['tracer'] = 'XXXX'

                    if not mounted_on == "Egis-tracker" and not mounted_on == "Fixed-rack":
                        module['mounted_on'] = 'Unknown-rack'
                        logger.error(f'The mounted_on key of {module.get("module_name")} must
                             be Fixed-rack or Egis-tracker')

                # OPETs are tracers that start with 'O'; these are the modules for OPETs only
                    , ignoring other tracers
                modules = [
                    x for x in config['modules']
                    if x['tracer'].startswith('O')
                ]

                # Add set_load_mode jobs
                for module in modules:
                    # check if module is enabled, and stopdate has yet not passed
                    module_enabled = not module.get('disabled', False)
                    module_not_expired = (
                        module.get('stopdate', None) is None
                        or datetime.datetime.strptime(module["stopdate"], "%Y-%m-%d").date()
                        >= datetime.datetime.now().date())

                    try:
                        if module_enabled and module_not_expired:
                            if module.get('load_mode'):
                                scheduled_time = present() + datetime.timedelta(seconds=3)
                                jobs[job_id] = {
                                    'scheduled_time': scheduled_time,
                                    'expiration_time': schedule_update_time,
```

```
                                    'opet_name': module['tracer'],
                                    'opet_bus': load_info[module['tracer']]['bus'],
                                    'opet_address': load_info[module['tracer']]['address'],
                                    'module_name': module['module_name'],
                                    'mounted_on': module['mounted_on'],
                                    'axis_azimuth': config[module['mounted_on']]['
                                        axis_azimuth'],
                                    'axis_tilt': config[module['mounted_on']]['axis_tilt'],
                                    'job_type': 'set_load_mode',
                                    'load_mode': module['load_mode'],
                                    'disabled':  module['disabled']
                                }
                                logger.debug(f'manager: {job_id} (set_load_mode: {module["
                                    load_mode"]}) scheduled for {scheduled_time.astimezone(
                                    TZ_LOCAL)}')
                                job_id += 1
                            else:
                                logger.error(f'Could not set load_mode job due to missing key
                                     at Tracer {module.get("tracer")}')

                        else: # Module should be disabled
                            scheduled_time = present()
                            jobs[job_id] = {
                                'scheduled_time': scheduled_time,
                                'expiration_time': schedule_update_time,
                                'opet_name': module['tracer'],
                                'opet_bus': load_info[module['tracer']]['bus'],
                                'opet_address': load_info[module['tracer']]['address'],
                                'module_name': module['module_name'],
                                'mounted_on': module['mounted_on'],
                                'axis_azimuth': config[module['mounted_on']]['axis_azimuth'],
                                'axis_tilt': config[module['mounted_on']]['axis_tilt'],
                                'job_type': 'set_load_mode',
                                'load_mode': 'disable',
                                'disabled':  module['disabled']
                            }
                            logger.debug(f'manager: {job_id} (set_load_mode: disabled)
                                scheduled for {scheduled_time.astimezone(TZ_LOCAL)}')
                            job_id += 1
                    except Exception:
                        logger.exception("Exception in scheduling set_load job")

                # Only schedule modules which are enabled
                modules = [
                    x for x in modules
                    if not x.get('disabled', False)
                ]

                # Only schedule modules which are not due
                modules = [
                    x for x in modules
                    if x.get("stopdate") is None or
                    datetime.datetime.strptime(x["stopdate"], "%Y-%m-%d").date() >= datetime.
                        datetime.now().date()
                ]

                # Time to update the jobs list
                if jobs:
                    schedule_start = max([job['scheduled_time'] for job in jobs.values()])
                else:
                    schedule_start = present()
                schedule_end = present() + datetime.timedelta(seconds=SCHEDULE_HORIZON)

                for module in modules:
                    # Schedule point measurements
                    point_interval = module.get('interval_point')


                    if point_interval is None or type(point_interval) is not int:
                        logger.error(f'Could not schedule point measurement since
                            interval_point is not set or not an integer for module {module.
```

```
                            get("module_name")}')
335
336                     else:
337                         point_interval = datetime.timedelta(seconds=point_interval)
338
339                         # Find times in the scheduling window, spaced by the interval
340                         scheduled_times = datetime_range(
341                             schedule_start,
342                             schedule_end,
343                             point_interval,
344                             include_start_point=False
345                         )
346
347                         # Jobs expire when they reach the scheduled time plus the
348                         # measurement interval, because then they are redundant with
349                         # the next repetition of the measurement.
350
351                         # Eliminate times that would already have expired
352                         scheduled_times = [
353                             t
354                             for t
355                             in scheduled_times
356                             if t + point_interval >= present()
357                         ]
358
359                         # Create a dict that gives the job instructions
360                         for scheduled_time in scheduled_times:
361                             jobs[job_id] = {
362                                 'scheduled_time': scheduled_time,
363                                 'expiration_time': scheduled_time + point_interval,
364                                 'opet_name': module['tracer'],
365                                 'opet_bus': load_info[module['tracer']]['bus'],
366                                 'opet_address': load_info[module['tracer']]['address'],
367                                 'module_name': module['module_name'],
368                                 'mounted_on': module['mounted_on'],
369                                 'axis_azimuth': config[module['mounted_on']]['axis_azimuth'],
370                                 'axis_tilt': config[module['mounted_on']]['axis_tilt'],
371                                 'job_type': 'point',
372                                 'data_destination': config['data_destination']
373                             }
374                             logger.debug(f'manager: {job_id} (point) scheduled for {
                                    scheduled_time.astimezone(TZ_LOCAL)}')
375                             job_id += 1
376
377                     # Schedule curve measurements
378                     curve_interval = module.get('interval_curve')
379
380                     if curve_interval is None or type(curve_interval) is not int:
381                         logger.error(f'Could not schedule curve measurement since
                                interval_curve is not set for module {module.get("module_name")}'
                                )
382
383                     else:
384                         curve_interval = datetime.timedelta(seconds=module.get('
                                interval_curve'))
385
386                         # Find times in the scheduling window, spaced by the interval
387                         scheduled_times = datetime_range(
388                             schedule_start,
389                             schedule_end,
390                             curve_interval,
391                             include_start_point=False
392                         )
393
394                         # Eliminate times that would already have expired
395                         scheduled_times = [
396                             t
397                             for t
398                             in scheduled_times
399                             if t + curve_interval >= present()
400                         ]
```

```
401
402                         # Create a dict that gives the job instructions
403                         for scheduled_time in scheduled_times:
404                             jobs[job_id] = {
405                                 'scheduled_time': scheduled_time,
406                                 'expiration_time': scheduled_time + curve_interval,
407                                 'opet_name': module['tracer'],
408                                 'opet_bus': load_info[module['tracer']]['bus'],
409                                 'opet_address': load_info[module['tracer']]['address'],
410                                 'module_name': module['module_name'],
411                                 'mounted_on': module['mounted_on'],
412                                 'axis_azimuth': config[module['mounted_on']]['axis_azimuth'],
413                                 'axis_tilt': config[module['mounted_on']]['axis_tilt'],
414                                 'job_type': 'curve',
415                                 'data_destination': config['data_destination']
416                             }
417                             logger.debug(f'manager: {job_id} (curve) scheduled for {
                                    scheduled_time.astimezone(TZ_LOCAL)}')
418                             job_id += 1
419
420                 # Remove jobs that have been waiting too long
421                 for old_job_id, old_job in list(jobs.items()):
422                     if present() - old_job["scheduled_time"] > MAX_JOB_TIME:
423                         logger.error(f'Removed job {old_job_id}: {old_job["job_type"]} for {
                                old_job.get("opet_name")}: too old')
424                         try:
425                             jobs.pop(old_job_id)
426                         except KeyError:
427                             pass
428
429                 logger.debug(f'manager: {len(jobs)} jobs are scheduled')
430                 logger.debug(f'manager: {jobs.keys()}')
431
432
433         # Handle Exceptions
434         except KeyboardInterrupt:
435             logger.error('Keyboard interrupt, shutting down child processes')
436             shutdown_event.set()
437         except Exception as e:
438             logger.exception(f'shutting down child processes; {e}')
439             shutdown_event.set()
440         finally:
441             shutdown_event.set()
442             # Wait for processes to finish
443             for process in processes:
444                 process.join(timeout=10)
445
446             # Otherwise terminate remaining processes
447             for process in processes:
448                 if process.is_alive():
449                     process.terminate()
450                     process.join(timeout=5)
```

D.2.2. Measurement loop

Code Listing D.14: Measurement loop

```
1  def measurement_loop(bus, jobs, jobs_in_progress, results, bus_info, load_info, maximum_wait,
       minimum_wait, shutdown_event, max_job_time):
2      """ Run scheduled measurement jobs for one OPET bus. This loop handles jobs assigned
3          to one bus and stores completed measurements in the shared results dictionary.
4
5      Args:
6          bus (str): Identifier of the OPET bus
7          jobs (DictProxy): Shared dictionary containing information of all scheduled jobs
8          jobs_in_progress (DictProxy): Shared dictionary containing the curve jobs that are in
               progress
9          results (DictProxy): Shared dictionary containg the results for the writer
10         bus_info (dict): Dictionary containing the adapter serial number connected to the bus
11         load_info (dict): Dictionary containing to which bus each OPET is connected to
12         maximum_wait (float): Maximum number of seconds into the future that
```

```
            it will wait for a job before returning to the main loop.
        minimum_wait (float): Minimum delay before returning the main loop
        shutdown_event (multiprocessing.Event): In case of shutting down this event is set
        max_job_time (datetime.timedelta): Maximum time a job should live after the schedule
            time
    """

    def initialize_bus():
        """Set up the OPET bus serial communication

        Returns:
            Tuple : a list of individual OPETs and
                the serial port. Returns None, None if something goes wrong.

        """

        try:
            serial_port_name = device_name(bus_info[bus]['adapter_serial_number'])[0]
            serial_port = Serial(serial_port_name, baudrate=200000, timeout=1)
            opet_bus = OPETBus(serial_port)
            this_bus_opet_addresses = [
                v['address']
                for k, v
                in load_info.items()
                if v['bus'] == bus
            ]
            opets = {
                address: OPET(opet_bus, address, iv_time_multiplier=2)
                for address
                in this_bus_opet_addresses
            }
            return opets, serial_port
        except IndexError:
            logging.error(f'Couldn\'t find the serial port for bus {bus}')
            return None, None
        except OPETTimeoutError:
            logging.error(f'Couldn\'t connect to OPET on bus {bus}')
            return None, None
        except UnexpectedReplyError:
            logging.error(f'Got unexpected reply while connecting to OPET on bus {bus}')
            return None, None
        except Exception as e:
            logging.error(f'Caught exception while connecting to OPET on bus {bus}: {e}')
            return None, None

    try:
        # Attempt to set up the bus
        opets, serial_port = initialize_bus()

        # Do jobs as they are scheduled and become due
        while not shutdown_event.is_set():
            if opets is None:
                # Attempt to set up the bus
                opets, serial_port = initialize_bus()
                # If it failed, wait before retrying
                if opets is None:
                    shutdown_event.wait(5)
                # Return to the top of the loop
                continue

            # These jobs belong to this bus, so other processes will not touch them
            bus_jobs = {
                job_id: job
                for job_id, job
                in jobs.items()
                if job['opet_bus'] == bus
            }

            # Consider each job that is scheduled for this bus
            # Sort by scheduled time
            bus_jobs = dict(sorted(bus_jobs.items(), key=lambda x: x[1]['scheduled_time']))
```

```
            this_bus_job_ids = list(bus_jobs.keys())

            # Repeat this as long as there are job ids listed for this bus
            while this_bus_job_ids:
                # Consider the first remaining job in this_bus_job_ids
                job_id = this_bus_job_ids[0]

                try:
                    job = jobs[job_id]
                except KeyError:
                    logger.error(f'bus {bus}: job {job_id}: job disappeared before processing
                        ')
                    this_bus_job_ids.pop(0)
                    continue




                # Determine how far into the future this job should be scheduled
                seconds_until_target = (
                    job['scheduled_time'] - present()
                ).total_seconds()

                # Ignore this job if it's too far into the future
                if seconds_until_target > maximum_wait:
                    # All other jobs are later than this, so this loop is finished
                    break

                # It's time to do this job. Pop it out of the shared jobs dict and
                # this bus's job id list.
                this_bus_job_ids.pop(0)

                try:
                    job = jobs.pop(job_id)
                except KeyError:
                    logger.warning(f'bus {bus}: job {job_id}: job disappeared before
                        execution')
                    continue

                # Delay until the right moment
                if shutdown_event.wait(max(0, seconds_until_target)):
                    break # If shutdown is set, break the loop

                # Check if this OPET is available:
                if not opets[job['opet_address']].available:
                    logger.debug(f'bus {bus}: job {job_id}: delayed because {job["opet_name
                        "]} is not available')
                    # The OPET isn't available (measuring a curve) so we'll
                    # put this job back on the shared dictionary, but not back in
                    # `this_bus_job_ids`, so we won't try again until the next loop
                    jobs[job_id] = job
                    # Return to the top of the loop to handle the next job in
                    # `this_bus_job_ids`
                    continue

                # Handle the job types differently
                # Set load mode, also enable output if disable is false
                if job['job_type'] == 'set_load_mode':
                    # mppt
                    if job["load_mode"] == 'mpp':
                        try:
                            opets[job['opet_address']].mode = 'mppt'
                            opets[job['opet_address']].output_enabled = True
                        except OPETTimeoutError:
                            logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in `
                                set_load_mode` on {job["opet_name"]}; Got an empty reply on
                                the OPET bus')
                        except UnexpectedReplyError:
                            logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in `
                                set_load_mode` on {job["opet_name"]}; Got an unexpected reply
                                 the OPET bus')
```

```
147                         except Exception as e:
148                             logger.error(f'bus {bus}: job {job_id}: Unexpected error in `
                                    set_load_mode` on {job["opet_name"]}; {e}')
149
150                     # voc
151                     elif job["load_mode"] == 'voc':
152                         try:
153                             opets[job['opet_address']].mode = 'voc'
154                             opets[job['opet_address']].output_enabled = True
155                         except OPETTimeoutError:
156                             logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in `
                                    set_load_mode` on {job["opet_name"]}; Got an empty reply on
                                    the OPET bus')
157                         except UnexpectedReplyError:
158                             logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in `
                                    set_load_mode` on {job["opet_name"]}; Got an unexpected reply
                                     the OPET bus')
159                         except Exception as e:
160                             logger.error(f'bus {bus}: job {job_id}: Unexpected error in `
                                    set_load_mode` on {job["opet_name"]}; {e}')
161
162                     # isc
163                     elif job["load_mode"] == 'isc':
164                         try:
165                             opets[job['opet_address']].mode = 'isc'
166                             opets[job['opet_address']].output_enabled = True
167                         except OPETTimeoutError:
168                             logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in `
                                    set_load_mode` on {job["opet_name"]}; Got an empty reply on
                                    the OPET bus')
169                         except UnexpectedReplyError:
170                             logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in `
                                    set_load_mode` on {job["opet_name"]}; Got an unexpected reply
                                     the OPET bus')
171                         except Exception as e:
172                             logger.error(f'bus {bus}: job {job_id}: Unexpected error in `
                                    set_load_mode` on {job["opet_name"]}; {e}')
173
174                     # disable output
175                     elif job["load_mode"] == 'disable':
176                         try:
177                             opets[job['opet_address']].output_enabled = False
178                         except OPETTimeoutError:
179                             logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in
                                    disable OPET on {job["opet_name"]}; Got an empty reply on the
                                     OPET bus')
180                         except UnexpectedReplyError:
181                             logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in `
                                    set_load_mode` on {job["opet_name"]}; Got an unexpected reply
                                     the OPET bus')
182                         except Exception as e:
183                             logger.error(f'bus {bus}: job {job_id}: Unexpected error in `
                                    set_load_mode` on {job["opet_name"]}; {e}')
184
185                 # Do point measurement
186                 elif job['job_type'] == 'point':
187                     # Point measurements are requested and recorded in immediate
188                     # succession
189
190                     # Do the job
191                     try:
192                         result = opets[job['opet_address']].sample
193                         logger.debug(f'bus {bus}: job {job_id}: {result["voltage"], result["
                                current"]}, {present() - job["scheduled_time"]} late')
194                         # Store the results
195                         results[job_id] = {
196                             'date_time': present(),
197                             'scheduled_time': job['scheduled_time'],
198                             'measurement_type': job['job_type'],
199                             'v': result['voltage'],
200                             'i': result['current'],
```

```
                            'temperature_cell': result['temperature_cell'],
                            'module_name': job['module_name'],
                            'mounted_on': job['mounted_on'],
                            'axis_azimuth': job['axis_azimuth'],
                            'axis_tilt': job['axis_tilt'],
                            'status_integer': result['status_integer'],
                            'data_destination': job['data_destination']
                        }
                    # We may get ValueError if the load returned something that
                    # isn't a valid float, for example
                    except ValueError:
                        logger.error(f'bus {bus}: job {job_id}: ValueError in `sample`
                            property on load {job["opet_name"]}; couldn\'t parse the load\'s
                            reply')
                        # The job has already been taken off the jobs list

                    #OPET may give error
                    except OPETTimeoutError:
                        logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in `sample`
                            property on load {job["opet_name"]}; Got an empty reply on the
                            OPET bus')
                    except UnexpectedReplyError:
                        logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in `
                            sample` property on load {job["opet_name"]}; Got an unexpected
                            reply the OPET bus')
                    except Exception as e:
                        logger.error(f'bus {bus}: job {job_id}: Unexpected error in `sample`
                            property on load  {job["opet_name"]}; {e}')

                elif job['job_type'] == 'curve':
                    # Curve measurements are first all requested as scheduled,
                    # then collected and recorded later in a separate step
                    # Request the curve
                    try:
                        reply = opets[job['opet_address']].start_iv_curve()
                        if reply:
                            logger.debug(f'bus {bus}: job {job_id}: curve measurement (start)
                                 {job_id}, {present() - job["scheduled_time"]} late')

                            # Add the job to jobs_in_progress
                            job['date_time'] = present()
                            job['measurement_duration'] = reply
                            jobs_in_progress[job_id] = job
                        else:
                            logger.debug(f'bus {bus}: job {job_id}: curve measurement (not
                                started) {job_id}, {present() - job["scheduled_time"]} late')
                    except OPETTimeoutError:
                        logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in
                            start_iv_curve property on load {job["opet_name"]}; Got an empty
                            reply on the OPET bus')
                    except UnexpectedReplyError:
                        logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in
                            start_iv_curve property on load {job["opet_name"]}; Got an
                            unexpected reply the OPET bus')
                    except Exception as e:
                        logger.error(f'bus {bus}: job {job_id}: Unexpected error in
                            start_iv_curve property on load {job["opet_name"]}; {e}')

            # Now check whether the jobs in progress are complete, reading back
            # results as they become available

            # These jobs belong to this bus, so other threads will not touch them
            bus_jobs_in_progress = {
                job_id: job
                for job_id, job
                in jobs_in_progress.items()
                if job['opet_bus'] == bus
            }
            # Sort by scheduled time so we can FIFO
            bus_jobs_in_progress = dict(sorted(bus_jobs_in_progress.items(), key=lambda x: x
                [1]['scheduled_time']))
```

```
257             for job_id, _ in bus_jobs_in_progress.items():
258                 # Pop a job out of the shared in-progress jobs dictionary
259                 try:
260                     job = jobs_in_progress.pop(job_id)
261                 except KeyError:
262                     logger.warning(f'bus {bus}: job {job_id}: in-progress job disappeared
                            before checking')
263                     continue
264                 # logger.debug(f'bus {bus}: job {job_id}: curve measurement (check if ready)
                        {present() - job["scheduled_time"]} late')
265                 # Check if the measurement is complete
266                 if not opets[job['opet_address']].available:
267                     measurement_complete = False
268                 else:
269                     try:
270                         measurement_complete = opets[job['opet_address']].operation_complete
                                ()
271                     except OPETTimeoutError:
272                         measurement_complete = False
273                         logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in check
                                operation complete status on {job["opet_name"]}; Got an empty
                                reply on the OPET bus')
274                     except UnexpectedReplyError:
275                         logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in check
                                 operation complete status on {job["opet_name"]}; Got an
                                unexpected reply the OPET bus')
276                         measurement_complete = False
277                     except Exception as e:
278                         logger.error(f'bus {bus}: job {job_id}: Unexpected error in check
                                operation complete status on {job["opet_name"]};{e}')
279                         measurement_complete = False
280
281                 if measurement_complete:
282                     # The measurement is ready
283                     logger.debug(f'bus {bus}: job {job_id}: curve measurement (complete) {
                            present() - job["scheduled_time"]} late')
284                     # Read the result
285                     try:
286                         result = opets[job['opet_address']].iv_data
287                         # Store the result
288                         results[job_id] = {
289                             'scheduled_time': job['scheduled_time'],
290                             'date_time': job['date_time'],
291                             'measurement_duration': job['measurement_duration'],
292                             'measurement_type': job['job_type'],
293                             'module_name': job['module_name'],
294                             'mounted_on': job['mounted_on'],
295                             'axis_azimuth': job['axis_azimuth'],
296                             'axis_tilt': job['axis_tilt'],
297                             'v': result['voltage'],
298                             'i': result['current'],
299                             'iv_status_integer': result['iv_status'],
300                             'temperature_cell': result['temperature_cell'],
301                             'data_destination': job['data_destination']
302                         }
303                     # We may get ValueError if the load returned something that
304                     # isn't a valid float, for example
305                     except ValueError:
306                         logger.error(f'bus {bus}: job {job_id}: ValueError in `iv_data`
                                property on load {job["opet_name"]}; couldn\'t parse the load\'s
                                reply')
307                         # The job has already been taken off the jobs_in_progress
308                         # list
309                     except OPETTimeoutError:
310                         logger.error(f'bus {bus}: job {job_id}: OPETTimeoutError in `iv_data`
                                 property on load {job["opet_name"]}; Got an empty reply on the
                                OPET bus')
311                     except UnexpectedReplyError:
312                         logger.error(f'bus {bus}: job {job_id}: UnexpectedReplyError in `
                                iv_data` property on load {job["opet_name"]}; Got an unexpected
                                reply the OPET bus')
```

```
                    except Exception as e:
                        logger.error(f'bus {bus}: job {job_id}: Unexpected error in `iv_data`
                            property on load {job["opet_name"]}; {e}')
                else:
                    # The measurement is not yet ready
                    # logger.debug(f'bus {bus}: job {job_id}: curve measurement (not yet
                        ready) {present() - job["scheduled_time"]} late')
                    # We popped this job out of the dict, now we put it back
                    jobs_in_progress[job_id] = job
                    continue
            shutdown_event.wait(minimum_wait)

    except Exception:
        logger.exception(f'bus {bus}: Exception in measurement_loop')
        raise
    finally:
        if serial_port is not None:
            try:
                serial_port.close()
                logger.info(f'bus {bus}: serial port closed')
            except Exception:
                logger.exception(f'bus {bus}: failed to close serial port')
```

D.2.3. Writer loop

Code Listing D.15: Writer loop

```
def writer_loop(results, data_path_base, tz_local, minimum_wait, shutdown_event):
    """Writer loop that writes the contents of results into the .csv file

    Args:
        results (DictProxy): contains all the recent results for storing
        dath_path (Path): Path to where the data should be stored
        TZ_LOCAL (ZoneInfo): Timezone information
        MINIMUM_WAIT (float): Infinite loops with nothing to do will delay this much before
            checking again
        shutdown_event (multiprocessing.Event): In case of shutting down this event is set
    """

    headers = {
        'point': [
            'date_time',
            'scheduled_time',
            'module_name',
            'mounted_on',
            'v',
            'i',
            'status_integer',
            'temperature_cell',
            'axis_azimuth',
            'axis_tilt'
        ],
        'curve': [
            'date_time',
            'scheduled_time',
            'measurement_duration',
            'module_name',
            'mounted_on',
            'v',
            'i',
            'iv_status_integer',
            'temperature_cell',
            'axis_azimuth',
            'axis_tilt',
        ]
    }

    try:
        while not shutdown_event.is_set() or results: # Write until no more results and
            shutdown is set
            while results:
```

```
                try:
                    result = results.pop(next(iter(results.keys())))
                except KeyError:
                    continue

                measurement_date = result['date_time'].astimezone(tz_local).date().isoformat
                    ()
                data_path = data_path_base / measurement_date / result['data_destination']
                data_file_path = (
                    data_path / (
                        'opet_results_'
                        + result['measurement_type']
                        + '_' + measurement_date
                        + '.csv'
                    )
                )

                # Convert datetimes to ISO 8601 strings
                result['date_time'] = result['date_time'].astimezone(tz_local).isoformat()
                result['scheduled_time'] = result['scheduled_time'].astimezone(tz_local).
                    isoformat()


                # Create the log file directory, if necessary
                data_file_path.parent.mkdir(parents=True, exist_ok=True)

                # Decide whether to write headers
                need_headers = not data_file_path.exists()
                with open(data_file_path, 'a', newline='') as log:
                    writer = csv.DictWriter(
                        log,
                        fieldnames=headers[result['measurement_type']],
                        extrasaction='ignore'
                    )
                    if need_headers:
                        writer.writeheader()
                    writer.writerow(result)
            shutdown_event.wait(minimum_wait)
    except Exception as e:
        logger.exception(f'writer_loop: exception error; {e}')
        raise
```