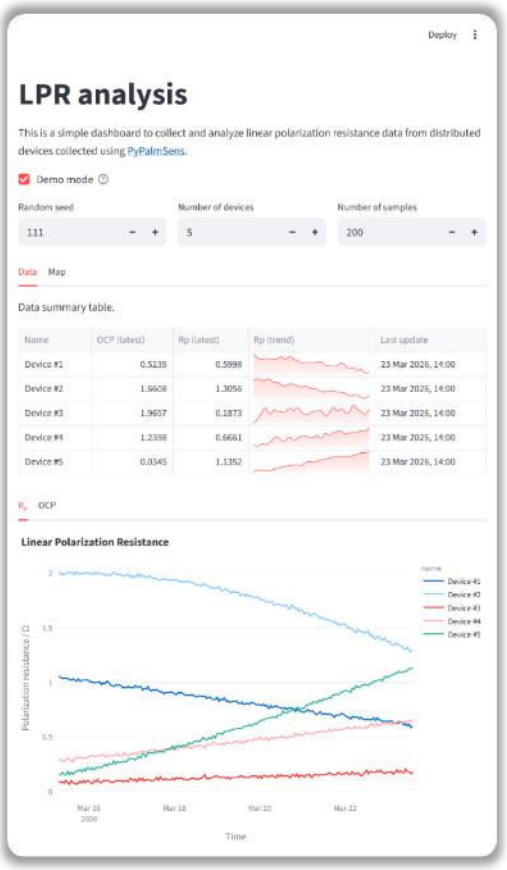
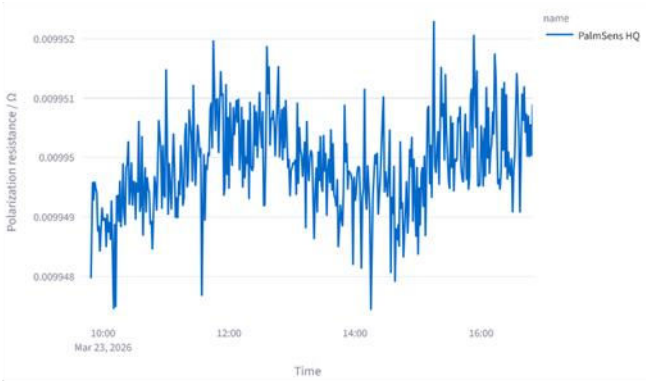
Remote IoT Corrosion Monitoring using Python





PalmSens

Application Note PSAN0102

www.palmsens.com

PSAN0102 | Last revision: April 2026

Application Note

1 Introduction

The global cost of corrosion is estimated to be US $2.5 trillion, which is equivalent to 3.4% of the global gross domestic product (GDP) (Koch et al. 2016). The economic loss can be divided across 5 specific industries: infrastructure, production and manufacturing, transportation, government, and utilities. Pipelines, bridges, and energy infrastructure all face continuous exposure to the environment, resulting in cracks, erosion, or other defects impacting safety and operational efficiency and eventually leading to failure or collapse.

Real-time monitoring for the effects of corrosion is therefore essential. The goal for this application note is to show how to set up a remote IoT system for corrosion sensing using a PalmSens potentiostat coupled to a low-power single-board computer for connectivity. The idea is to have multiple of these 'in the field' attached to wind turbines, bridges, or other corroding infrastructure and monitor them via a central dashboard. The dashboard can be used to monitor corrosion rate over time.

The application note consists of 3 parts:

1. Measurement script to record the corrosion data using Python
2. Setup for one or more Raspberry Pi controllers to collect data at regular intervals
3. Dashboard to collect, process, and display data from multiple Raspberry Pi's

We have chosen Raspberry Pi controllers for this application note, because it is a well-known, low-price, widely available single-board computer. Any other single-board computer or PC will also work, as long as it runs Python and supports PyPalmSens.

All the code reproduce this application note is available from our code repository.

2 What is Linear Polarization Resistance?

Linear Polarization Resistance (LPR) is an electrochemical technique used to monitor corrosion rates and investigate the corrosion behavior of metals in a conductive environment. LPR is a quick and non-destructive technique that is similar to Linear Sweep Voltammetry (LSV).

The LPR method consists of performing a potential sweep over typically 5-15 mV around the corrosion potential E_{corr}, which is equivalent to the open-circuit potential (OCP). This induces a current between the working and counter electrodes. Note that the potential range must be small enough so that the induced current response is linear. This varies with each material and analytical setup.

The polarization resistance R_p is defined as the change in the resistance over the applied voltage delta.

$$R_p = \frac{\Delta E}{\Delta I}$$

Over a small potential range, R_p is therefore constant and can be found by fitting a linear regression to the potential vs. current curve, where the slope corresponds to R_p. According to the Stern–Geary equation (Stern and Geary 1957), R_p is inversely proportional to the instantaneous corrosion rate under certain conditions.

$$R_p = \frac{1}{I_{corr}} \cdot \frac{\beta_a \beta_c}{2.3\ (\beta_a + \beta_c)}$$

Where I_{corr} is the corrosion current and β_a and β_c are the anodic and cathodic Tafel constants. These constants are largely determined by the nature of the material (electrode/metal) and the corrosive medium (electrolyte/environment). Finally, the I_{corr} can be used to determine the corrosion rate in mm/year according to equation:

$$Corrosion\ Rate = I_{corr}\ \frac{K \cdot EW}{\rho \cdot A}$$

Where K is a constant defined by ASTM G102 (1989) as $3272\ mm/(A\ cm\ year\ mol)$ and EW is the equivalent weight of the corroding material (the atomic weight divided by the number of electrons needed for conversion).

3 Measuring LPR using Python

To collect LPR data, we use the LSV method in PyPalmSens. PyPalmSens is a free and open-source Python library for automating electrochemistry experiments using PalmSens instruments. We used the experimental parameters described in this knowledge base article.

As our goal is to demonstrate the software, we have chosen parameters for a quick measurement rather than those used for real samples. We set the begin and end potential to -0.1 V and 0.1 V, with 5 mV steps and a scan rate of 20 V / s. E_{corr} is more generally referred to as the OCP, so to measure around E_{corr}, we set the versus OCP `mode` to 3. This sets a flag that the *begin* and *end* potential are relative to the OCP. The `max_ocp_time` is set to 5 s. This corresponds to the time allowed for the system to reach a stable OCP. For relatively stable processes like corrosion, 5 s is usually more than enough.

```
import pypalmsens as ps

method = ps.LinearSweepVoltammetry(
    begin_potential = -0.1,
    end_potential = 0.1,
    step_potential = 0.005,
    scanrate = 0.02,
    versus_ocp = {
      'max_ocp_time': 5.0,
      'mode': 3,
    }
)
measurement = ps.measure(method)
```

After the measurement, we can retrieve the E_{corr} / OCP (in V) and measurement timestamp (in ISO 8601 format) via their attributes:

```
print(f'ocp: {measurement.ocp_value} V')
# ocp: 1.6056 V
print(f'time: {measurement.timestamp}')
# time: 2026-03-16T11:51:01
```

Obtain the polarization resistance by fitting a linear regression model to the change in potential vs current curve using the polynomial module from the numpy library. We store the polarization resistance as defined by the slope of the linear fit in the variable `rp`.

```
import numpy as np

current = measurement.dataset["Current"]
```

```
potential = measurement.dataset["Potential"]

ffit = np.polynomial.Polynomial.fit(potential, current, 1).convert()

intercept, slope = ffit.coef
rp = slope

print(f'Rp: {rp} Ohm')
# Rp: 0.00995 Ohm
```

The complete code example can be found in the file measure.py on our code repository.

4 Setup Raspberry Pi

For this application note, we used a Raspberry Pi 4 model B controller. We used the Raspberry Pi Imager to install Raspbian (based on Debian 13 "trixie") on a 16 GB sd card. See the Raspberry Pi software page for specific installation instructions. If you use another (single-board) computer, or a different Linux distribution, keep in mind that the steps may be (slightly) different.

The next sections show how to:

1. Set up the measurement
2. Set up webserver
3. Set up recurring measurement task

Although you can connect a monitor, keyboard, and mouse to the Raspberry Pi and follow the instructions that way, we used SSH to remotely control the Raspberry Pi from our PC over the network. First, we need to enable the SSH server, which is disabled by default. You can either enable SSH while installing Raspbian or afterwards. See the Raspberry Pi documentation for further instructions. Make a note of the IP address, so that you can log into the Raspberry Pi device. The IP address for our Raspberry Pi device is 192.168.1.115. Log in using the SSH command:

```
ssh pi@192.168.1.115
```

When it asks for a password, enter the default password: 'raspberry'. If you set up your system with a different user or password, use that instead.

Once you are logged into the Raspberry Pi via SSH, continue with the following steps.

4.1 Prerequisites

There are a few steps to make sure your Raspberry Pi can run PyPalmSens. For complete instructions, see the PyPalmSens installation page. In summary:

1. Install the .NET runtime 9.0 or newer following the steps described here. Verify that `dotnet` is installed and `DOTNET_ROOT` is set:

```
dotnet --version
# 9.0.306
echo $DOTNET_ROOT
# /home/pi/.dotnet
```

2. Make sure your user is added to the `dialout` group:

```
groups
# pi adm dialout ...
```

If your username is not among the list, add it using:

```
sudo usermod -a -G dialout $USER
```

3. If you use an FTDI device like the EmStat Pico, you also need to install the FTDI drivers. For more information, see this link.

4.2 Set up the webserver

There are many options to share data over the network. For this application note, we choose to make the data available via a simple http webserver. We use apache2, because it is the most popular webserver on the internet and easy to get started with.

To install, type the following commands in your shell to install apache2 and start the webserver:

```
sudo apt install apache2
sudo service apache2 start
```

Open http://192.168.1.115/ and you should see the apache2 welcome page (replace with IP of your Raspberry Pi device). By default, apache2 listens on port 80 to serve the data.

Any data placed in the `/var/www/html` directory will be made available on the network. After first installation, this is restricted to the root user. Add your user to the `www-data` group and modify the permissions so that the logged-in user can also write to the `/var/www/html` directory. Restart the server for the changes to take effect.[1]

```
sudo usermod -aG www-data $USER
sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 775 /var/www/html
sudo service apache2 restart
```

4.3 Set up the measurement

With the server in place, we can set up the environment to run the measurement script above. Create a virtual Python environment and install dependencies:

```
mkdir ~/lpr && cd lpr
python -m venv .venv && source .venv/bin/activate
pip install pypalmsens==1.8.0 numpy
```

This creates a new directory `lpr`, a python environment in `.venv` and installs the needed dependencies. Download the measurement script to the `~/lpr` directory:

```
wget https://raw.githubusercontent.com/PalmSens/PalmSens_SDK/refs/heads/main/application_notes/remote_corrosion/measure.py
```

At this stage you can test whether the script works. If you have a device connected this should return the date and OK.

[1] https://superuser.com/questions/1537910/write-permission-in-var-www-html-for-my-user

```
python measure.py
# 2026-03-27T15:14:31 OK
```

A `data.csv` file will be placed in `/var/www/html` and served on the network at http://192.168.1.115/data.csv (substitute your RPi's IP).

4.4 Set up recurring task

To make sure the measurement script is repeated at regular intervals, you can use a tool like cron. Cron is a simple tool that is widely used to set up recurring tasks on Linux systems. It is a time-based job scheduler and can be used to automate any task that can run from the command line. The cron syntax is known for its conciseness and flexibility, but it can be a bit tricky at first. We recommend using a tool like cron.help to get started if you are unfamiliar with the tool.

First, ensure that cron is running:

```
sudo service cron status
```

Enable cron:

```
sudo service cron start
```

Type `crontab -e` to edit the cron tables. Add the following two lines:

```
DOTNET_ROOT=/home/pi/.dotnet
*/1 * * * * cd /home/pi/lpr && source .venv/bin/activate && python measure.py >>
measure.log 2>&1
```

Note that the second line runs all the way to the end (no line break). The `*/1 * * * *` instruction tells cron to schedule this job every minute. This is convenient for testing purposes. Later, you can replace it with the required schedule, e.g. `0 * * * *` to run at the start of every hour.

The job changes directory to `/home/pi/lpr/`, activates the virtual environment and starts the measurement script. Output is logged to `measure.log`. On a successful job, this should show something like `2026-03-17T16:12:05 OK` after each measurement.

Note that we set the `DOTNET_ROOT` environment variable in crontab. This is necessary for PyPalmSens to find the dotnet libraries that it is based on. If dotnet was installed in another location, find the correct path via the environment variable:

```
echo $DOTNET_ROOT
# /home/pi/.dotnet
```

On our system this returns `/home/pi/.dotnet`.

Once this step is done, the data will be served on the network http://192.168.1.115/data.csv (substitute your RPi's IP). Any subsequent measurement will append to this file.

4.5 A note on security

Our goal for this application note was to keep the setup simple, using familiar tools that require little setup.

We used SSH to login to the Raspberry Pi controls using the default port (`22`) and password (`pi` / `raspberry`). We also used apache2 to set up a webserver and share unencrypted data over the network. While the network setup here is fine for a trusted local network, without additional configuration this is not secure for production use.

Please consult a networking expert before exposing your Raspberry Pi to the internet.

See for example these guides:

- https://linuxize.com/post/ssh-hardening-best-practices/
- https://httpd.apache.org/docs/2.4/misc/security_tips.html

5 Monitoring dashboard

To monitor the remote data, we want to have some sort of central interface to retrieve, display, and analyze the data, without requiring any code.

Dashboards were introduced in the late 2010s, coming from the data science world making use of the modern web. In essence, dashboards are simple web-apps used to quickly glance at some data. The limited scope means they are typically fast to develop or put together. They are helpful when the intent is to share reproducible charts and data with a non-technical audience.

If you are using Python, these are the ones you should be looking at:

- Plotly/Dash (2017, 8M downloads/month)
- Panel (2018, 3M downloads/month)
- Streamlit (2019, 32M downloads/month)
- Jupyter + Voilà (2019, 16M downloads/month)

If you are looking outside the Python ecosystem, there are many more options for setting up visualizations, the most well-known ones being Tableau and PowerBI. Since the data are published using CSV (or whatever data format fits your use case), you can use any tool to ingest and represent the data.

We chose Streamlit to develop the dashboard for this application note. We find Streamlit convenient for prototyping. The API is well-designed, it looks good out-of-the-box, and it works with all the popular data and plotting tools (plotly, matplotlib, etc). The linear execution model makes it easy to reason about the code, because there is no need for any callbacks or complex flow controls.

To retrieve the data, we used the `pd.read_csv()` function from the pandas library. This makes working with csv data straightforward, because `read_csv()` can directly load data from either a local path or url. Pandas works well in combination with Steamlit. The data table was designed to work with dataframes directly. The plots were generated with plotly.

You can find the complete example dashboard script `app.py` via our code repository. install the dependencies using:

```
pip install --requirements-from-script app.py
```

And start the dashboard using:

```
streamlit run app.py
```

This will open the dashboard in your web browser or show the IP address that it is hosted at. See the Streamlit documentation for more information on how to configure and modify the dashboard.

Figure 1a shows an overview of the dashboard. For testing, it has a function to generate semi-random data. The overview table shows the device name, the latest OCP and R_p value, and a trend line showing the latest measured points.

Figure 1. (a) Overview of the dashboard with semi-random generated mock data. The overview shows a table summarizing the data, and R_p or OCP charts. (b) Click on the 'Map' tab to show device locations and data on a map. Select points on the map to display the corresponding data in the R_p/OCP chart. (c) Example LPR data measured on an EmStat Pico using a dummy cell over the course of several days.

If you are working with your own data (uncheck demo data), new devices can be added by appending a new row to the table with the device name, GPS location, and data path (Figure 2). For many devices and routine use, it may be more pragmatic to change the code to load the devices from a config file.

Clicking on the 'Map' tab, shows the device map (Figure 1b). This shows the location of each device with the color encoding the latest R_p value. The charts below show the R_p and OCP data.

name	lon	lat	path
PalmSens HQ	5.1497	52.0202	http://192.168.1.115/data.csv
My location	1.123	6.789	http://192.168.1.123

Figure 2. Input new devices by adding new rows to the table.

6 Conclusion

With this application note we showed how to set up a remote IoT system for corrosion monitoring, using a central dashboard to retrieve and display the data. The LPR measurement script is central to this application note. We used PyPalmSens to show how to set up a linear sweep voltammetry method to measure the polarization resistance using a few lines of code.

We purposefully kept the scope for this application note small, using familiar tools, standard protocols, and simple data structures. That said, we see many opportunities to scale up the example set up, such as scaling up to multiple devices, improving the data management, or increasing the frequency for real-time monitoring.

Additional devices can be added by simply repeating the setup steps. Once you have a setup that fits the requirements, you can simply clone the SD card. Alternatively, you can use a software provisioning and configuration management tool like Ansible to manage devices. The example dashboard already allows for adding multiple devices or data files.

The data were exported to csv files, which makes it straightforward to import in any data analysis workflow. The dashboard uses the pull method to retrieve the data over http. The data could instead be stored in a local database exposed over the network to enable better querying of the data and additional security. Alternatively, this model could be flipped, where the devices push new measurements directly to a central database.

We implemented our dashboard in based on open-source tools in the Python ecosystem. Here too, there are many alternative options as discussed above.

The full code for the measurement script and dashboard, including sample data, are available from our code repository.

7 References

- ASTM International. 1989. *Standard Practice for Calculation of Corrosion Rates and Related Information from Electrochemical Measurements*. G102-89(2015)e1. doi:10.1520/G0102-89R15E01.
- Koch, Gerhardus, Jeff Varney, Neil Thompson, Oliver Moghissi, Melissa Gould, and Joe Payer. 2016. *International Measures of Prevention, Application, and Economics of Corrosion Technologies Study*. NACE International. http://impact.nace.org/documents/Nace-International-Report.pdf.
- Stern, M., and A. L. Geary. 1957. “Electrochemical Polarization: I . A Theoretical Analysis of the Shape of Polarization Curves.” *Journal of The Electrochemical Society* 104(1):56. doi:10.1149/1.2428496.