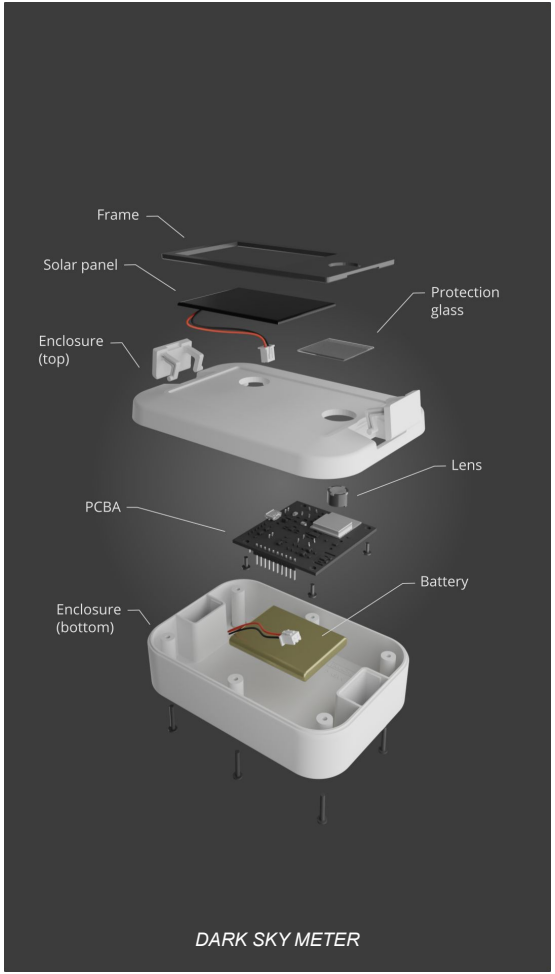
Using embedded Rust to build an unattended, battery-powered device

FOSDEM 2025 / Brussels
Embedded, Mobile and Automotive

SC Robotics



DISCLAIMER: This is **not a deep dive into Rust** but a series of thoughts about its feasibility for developing production-ready embedded devices

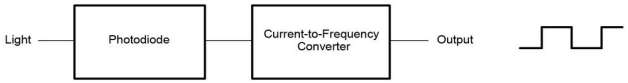
Light pollution is a big problem for astronomers

By Jeremy Stanley - originally posted to Flickr as Light pollution: It's not pretty, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=5680800

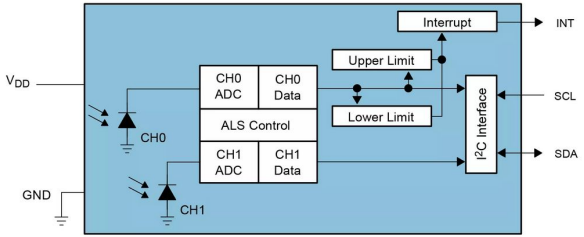
Other consequences

1. Public health
2. Ecological
3. *Noctalgia* / Cultural identity

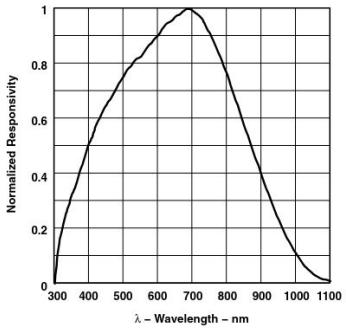
How to measure light pollution?

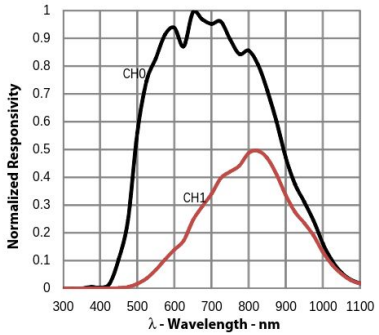


ams TSL237 (discontinued)

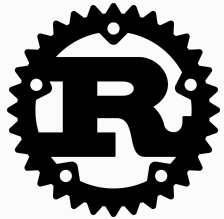


ams TSL25911



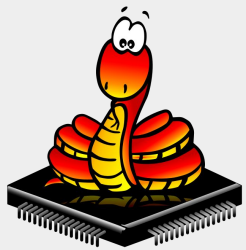


Trends with impact in the industry
(Rust and Zephyr)



Traditional approach
(C/C++ with vendor's SDKs)

Popular among Maker Community
(Arduino, MicroPython/CircuitPython)



Why Rust?

Powerful static analysis

Enforce pin and peripheral configuration at compile time. Guarantee that resources won't be used by unintended parts of your application.

LEARN MORE

Flexible memory

Dynamic memory allocation is optional. Use a global allocator and dynamic data structures. Or leave out the heap altogether and statically allocate everything.

LEARN MORE

Fearless concurrency

Rust makes it impossible to accidentally share state between threads. Use any concurrency approach you like, and you'll still get Rust's strong guarantees.

LEARN MORE

Source: https://www.rust-lang.org/what/embedded

Why Rust?

Powerful static analysis

Enforce pin and peripheral configuration at compile time. Guarantee that resources won't be used by unintended parts of your application.

LEARN MORE

```rust
struct GpioOutput;
struct GpioInput;

struct Gpio<Mode> {
    status: bool,
    mode: Mode,
}

impl Gpio<GpioInput> {
    fn read(&self) -> bool {
        self.status // Read the value (input)
    }

    fn into_output(self) -> Gpio<GpioOutput> {
        Gpio {
            status: self.status,
            mode: GpioOutput,
        }
    }
}

impl Gpio<GpioOutput> {
    fn write(&mut self, status: bool) {
        self.status = status; // Set the value (output)
    }

    fn into_input(self) -> Gpio<GpioInput> {
        Gpio {
            status: self.status,
            mode: GpioInput,
        }
    }
}
```

Fearless concurrency

akes it impossible to accidentally
state between threads. Use any
rency approach you like, and you'll
t Rust's strong guarantees.

LEARN MORE

Source: https://www.rust-lang.org/what/embedded

Why Rust?

Powerful static analysis

Enforce pin and peripheral configuration at compile time. Guarantee that resources won't be used by unintended parts of your application.

LEARN MORE

```
struct GpioOutput;
struct GpioInput;

struct Gpio<Mode> {
    status: bool,
    mode: Mode,
}

impl Gpio<GpioInput> {
    fn read(&self) -> bool {
        self.status // Read the value (input)
    }

    fn into_output(self) -> Gpio<GpioOutput> {
        Gpio {
            status: self.status,
            mode: GpioOutput,
        }
    }
}

impl Gpio<GpioOutput> {
    fn write(&mut self, status: bool) {
        self.status = status; // Set the value (output)
    }

    fn into_input(self) -> Gpio<GpioInput> {
        Gpio {
            status: self.status,
            mode: GpioInput,
        }
    }
}
```

Fearless concurrency

akes it impossible to accidentally
state between threads. Use any
rency approach you like, and you'll

```
fn main() {
    let gpio = Gpio {
        status: false,
        mode: GpioInput,
    };

    let status = gpio.read();
    let _ = gpio.write(true); // no method named `write`

    let gpio = gpio.into_output();
    gpio.write(true);
}
```

Why Rust?

```rust
#![no_std]

use heapless::Vec as HeaplessVec;
use std::vec::Vec;

fn main() {
    // Does not compile in no_std!!!
    let vec: Vec<u8> = Vec::<u8>::new();

    // Use custom allocator instead
    let hvec = HeaplessVec::<u8, 8>::new();
}
```

Flexible memory

Dynamic memory allocation is optional. Use a global allocator and dynamic data structures. Or leave out the heap altogether and statically allocate everything.

LEARN MORE

Fearless concurrency

Rust makes it impossible to accidentally share state between threads. Use any concurrency approach you like, and you'll still get Rust's strong guarantees.

LEARN MORE

Source: https://www.rust-lang.org/what/embedded

Why Rust?

Source: https://www.rust-lang.org/what/embedded

State-of-the-Art tooling at last!

Do you want to cross build the project?

```
$ cargo build
```

Do you want to upload the project to the board?

```
$ cargo run
```

Do you want to cross build the project?

```
$ cargo test
```

Do you want to add a new functionality?

```
$ cargo add postcard
```

Thousand of interesting crates available (e.g. serde, defmt, bitfields, heapless, smol, svd2rust, smoltcp...)

What about embedded systems?

Do you want to add a new sensor?

```
$ cargo add bme280
```

Hundreds of drivers available thanks to the standardization effort of the Rust Embedded Devices Working Group

```rust
let i2c_config = I2cConfig::new().baudrate(100.kHz().into());
let sda = peripherals.pins.gpio5;
let scl = peripherals.pins.gpio6;
let i2c = I2cDriver::new(peripherals.i2c0, sda, scl, &i2c_config).unwrap();

let i2c_ref_cell = core::cell::RefCell::new(i2c);
```

→ `$ cargo add bme280`

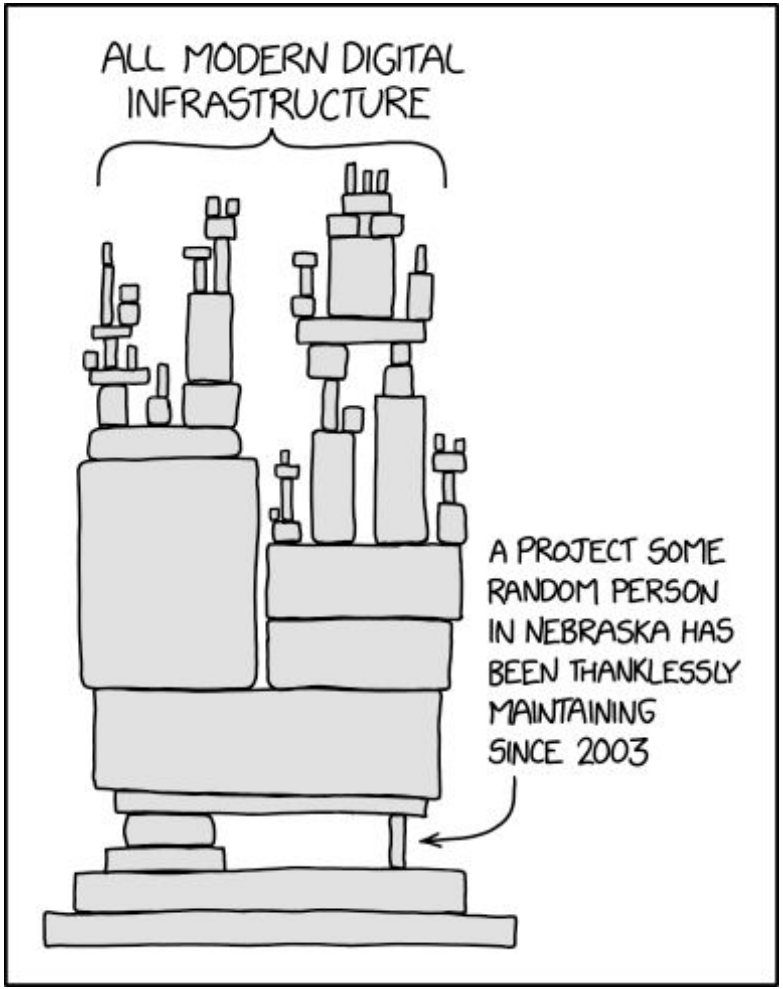
```rust
{
    let mut bme280 = bme280::i2c::BME280::new_secondary(
        embedded_hal_bus::i2c::RefCellDevice::new(&i2c_ref_cell),
    );
    bme280.init(&mut delay::FreeRtos).unwrap();

    let data = bme280.measure(&mut delay::FreeRtos).unwrap();
    info!(
        "BME280: {:?}",
        (data.temperature, data.humidity, data.pressure)
    );
}
```

→ `$ cargo add adxl345-eh-driver`

```rust
{
    let mut adxl345 = adxl345_eh_driver::Driver::new(
        embedded_hal_bus::i2c::RefCellDevice::new(&i2c_ref_cell),
        Some(adxl345_eh_driver::address::SECONDARY),
    )
    .unwrap();

    let data = adxl345.get_accel().unwrap();
    info!("ADXL345: {:?}", data);
}
```



Modern tools,
modern problems!

Considering Rust for your new project?

✅ A strong and cohesive Community

✅ State-of-the art tooling

❕ Decent documentation

❌ Vendor support

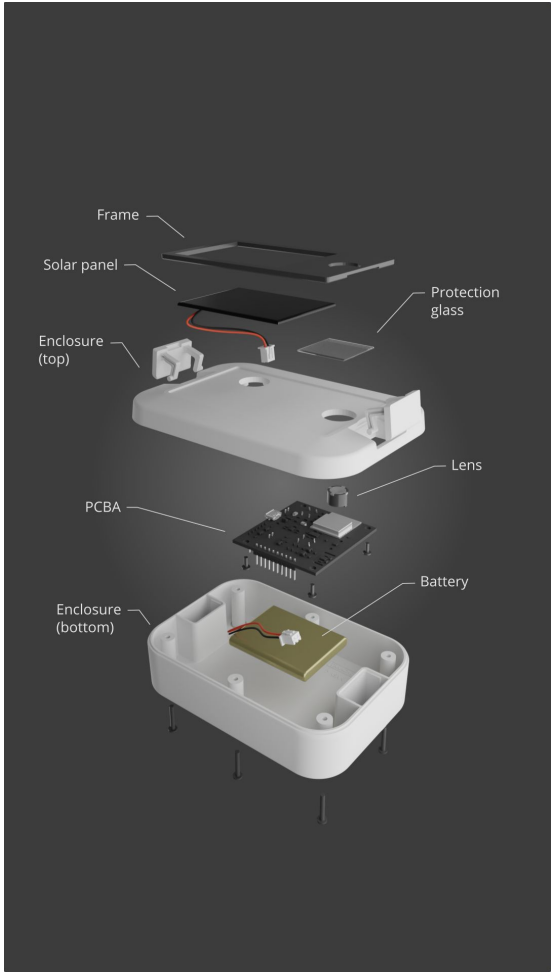
Dark Sky Meter Features

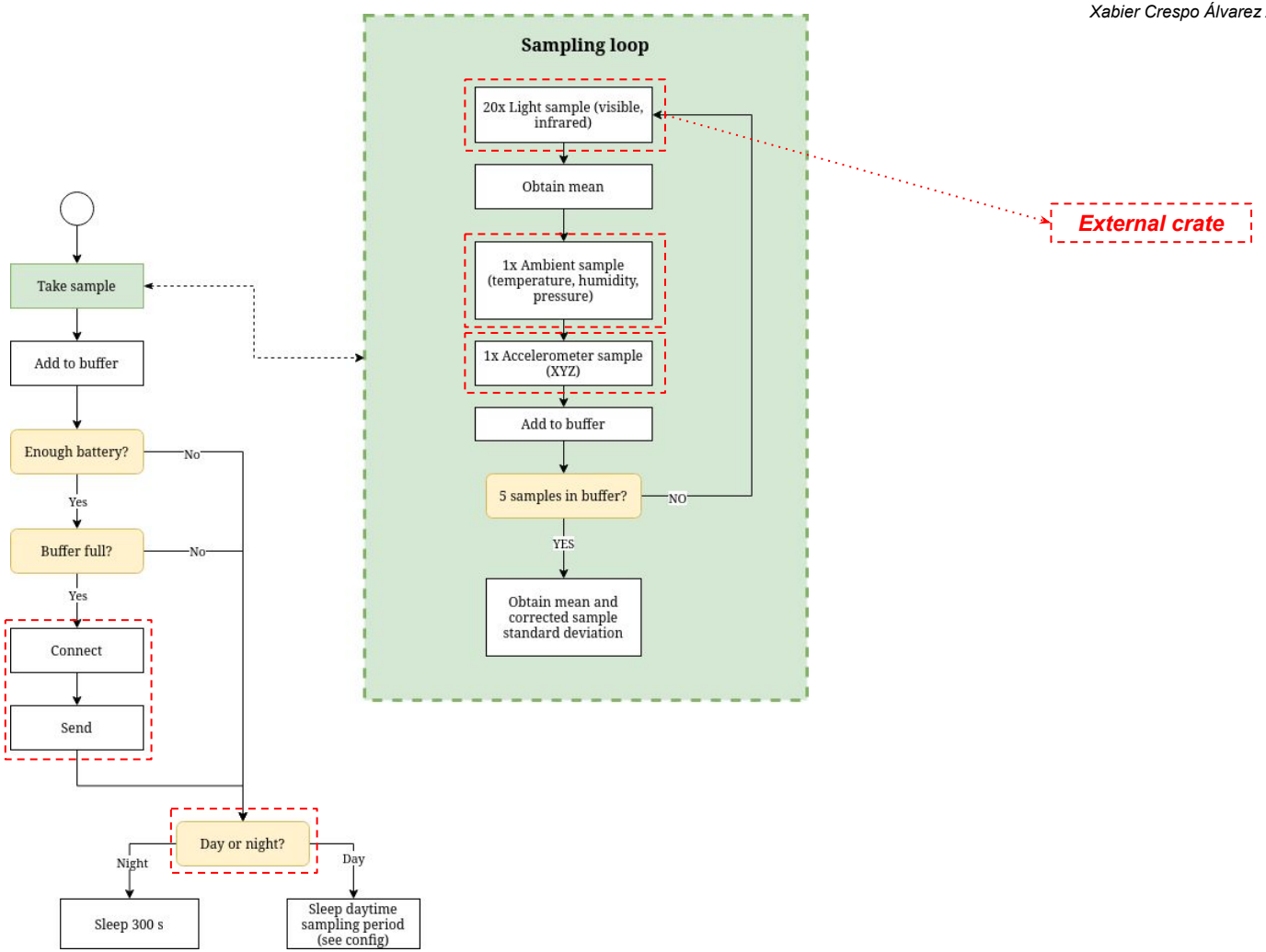
- Sensors:
  - Measures light intensity (TSL2591)
  - Measures *ambient* (inside the enclosure) temperature, humidity and pressure (BME280)
  - Measures acceleration –and thus pointing- (ADXL345)

- Connectivity options:
  - WiFi (ESP32-C3) <---- also the main MCU
  - Cellular, LTE-M with 2G fallback (Quectel BG95)
  - LoRaWAN, compliant with spec 1.0.3 specification (RAK3172)

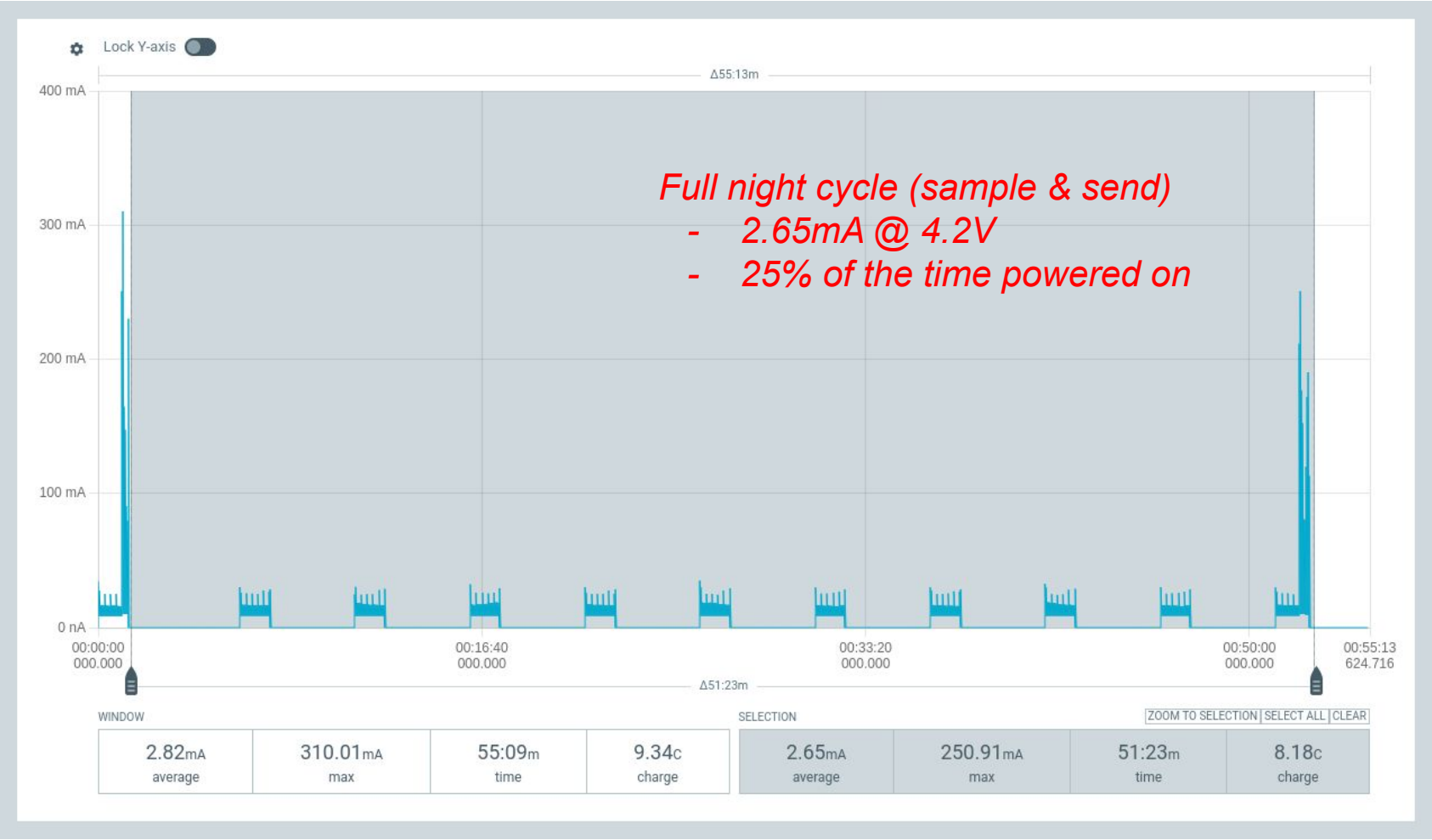
- User configurator available through WiFi AP
- 1x buzzer, 1x RESET button and 1x USER button
- Powered by solar panels and batteries

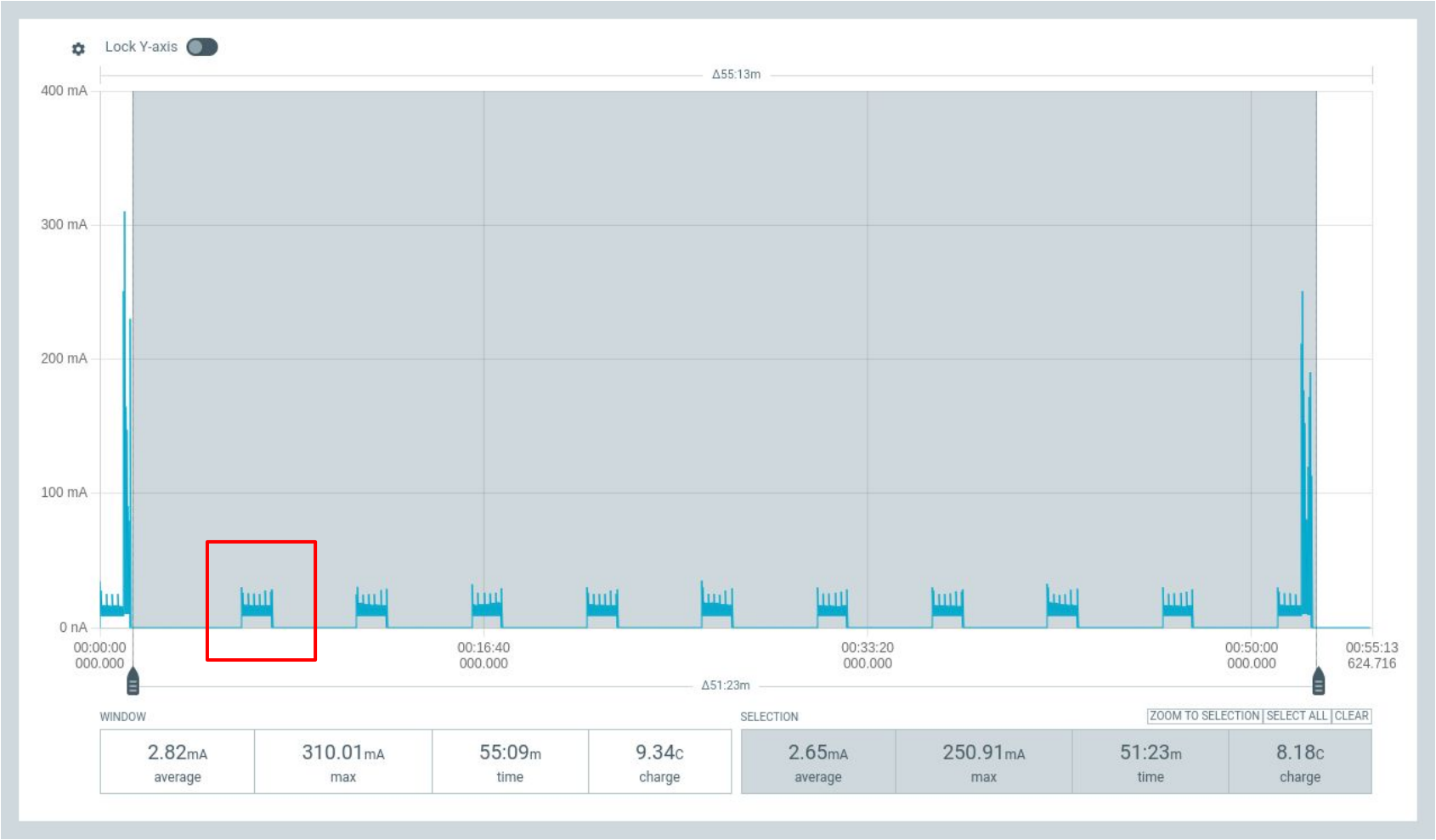
- Low power software features:
  - Stops using networking if voltage is too low, resumes operation once battery is charged
  - Stores data in RAM, only to be sent in groups
  - Capable of using different sampling rates at day and night

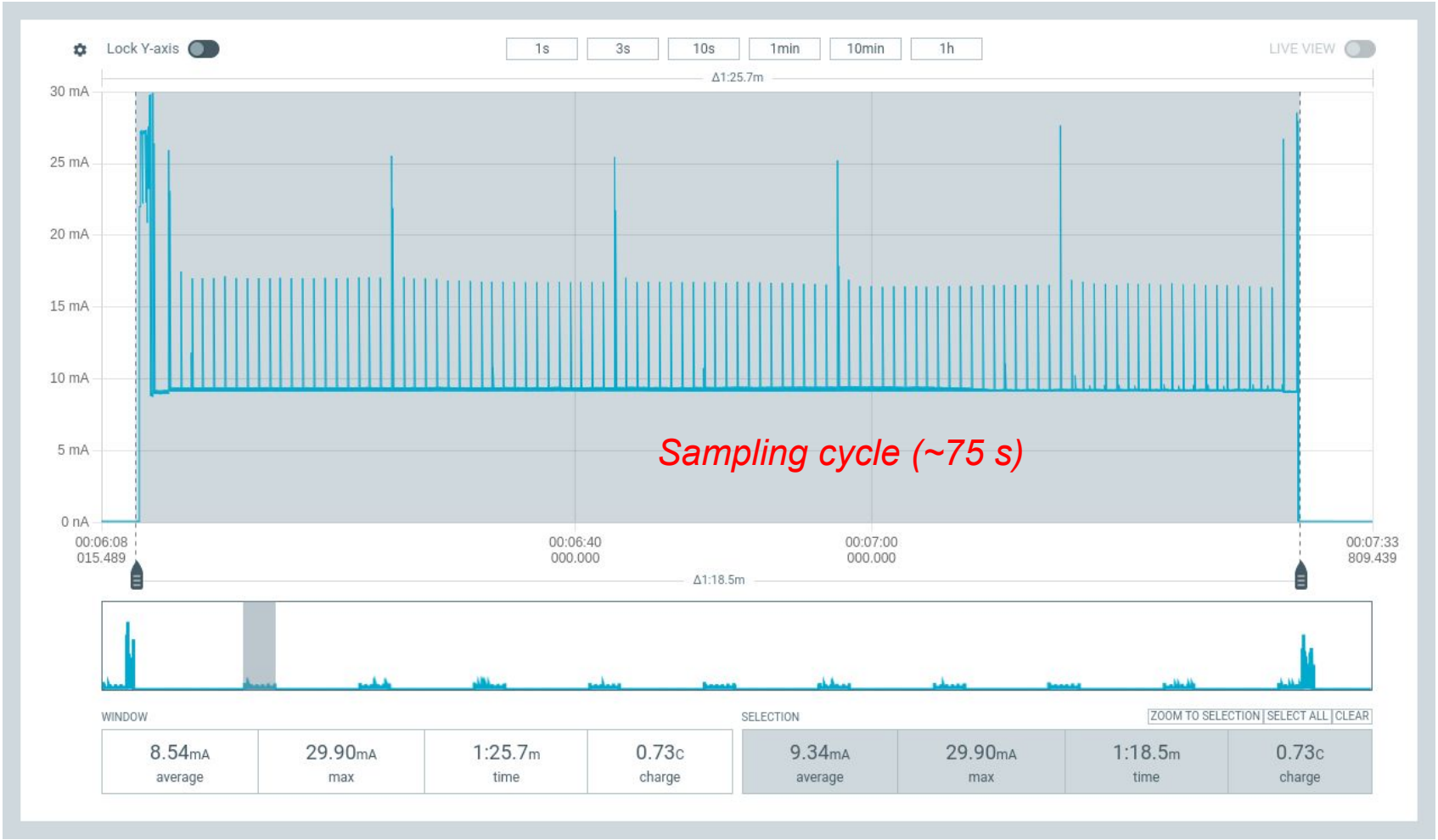


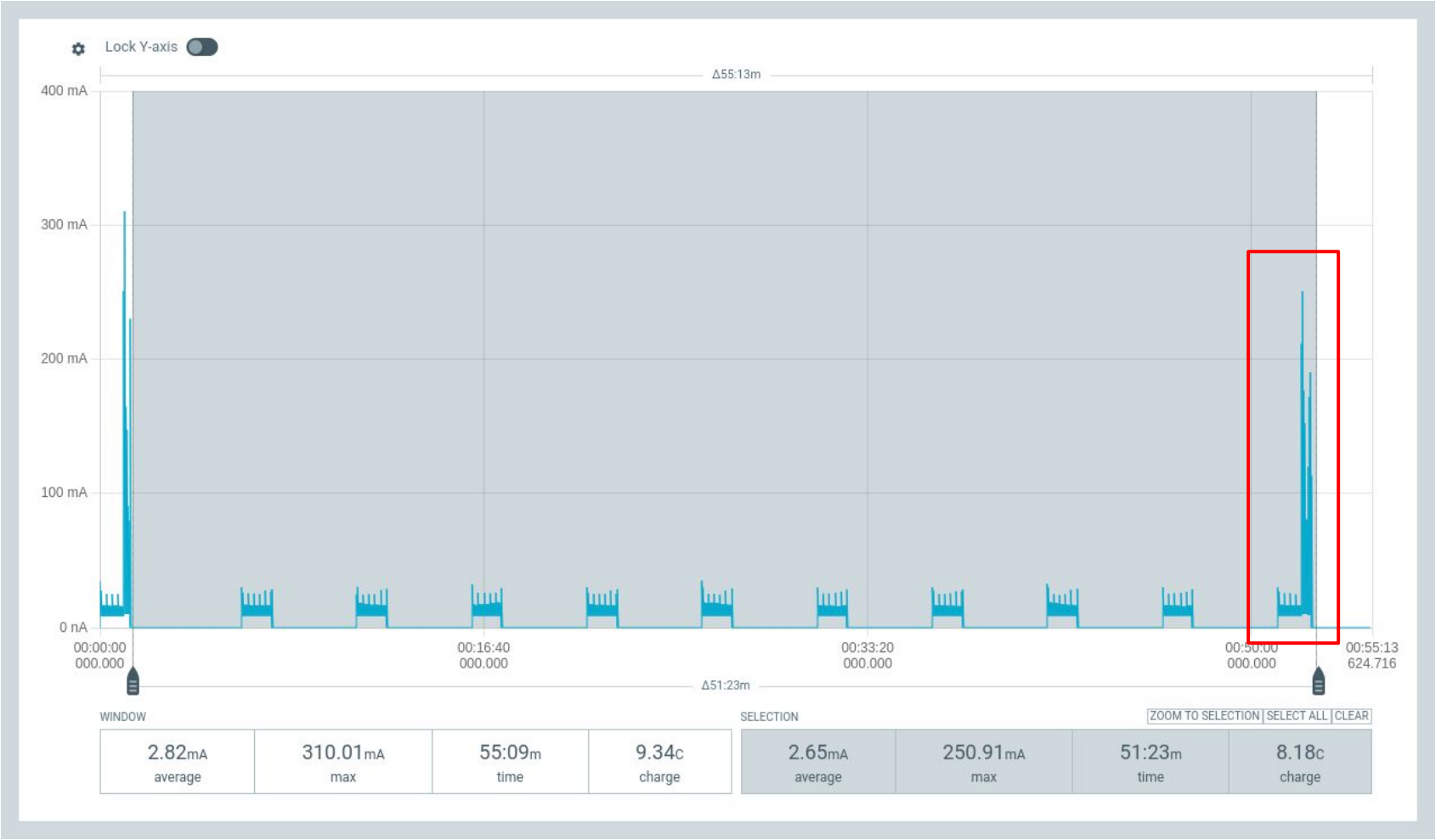

Sampling loop
20x Light sample (visible, infrared)
Obtain mean
1x Ambient sample (temperature, humidity, pressure)
1x Accelerometer sample (XYZ)
Add to buffer
5 samples in buffer?
NO
YES
Obtain mean and corrected sample standard deviation
External crate
Take sample
Add to buffer
Enough battery?
No
Yes
Buffer full?
No
Yes
Connect
Send
Day or night?
Night
Day
Sleep 300 s
Sleep daytime sampling period (see config)

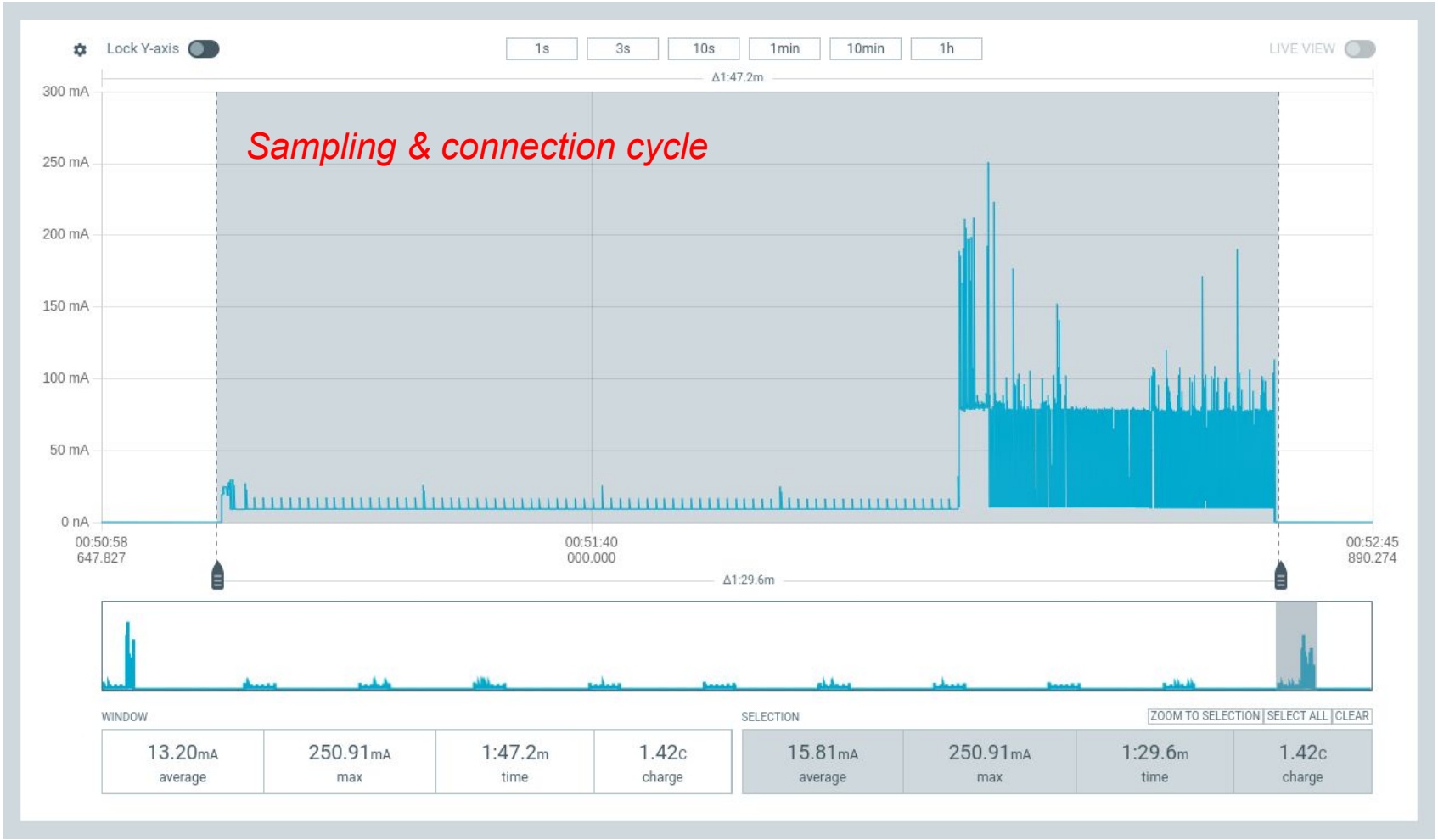
POWER CONSUMPTION


Lock Y-axis
Δ55:13m
400 mA
300 mA
200 mA
100 mA
0 nA
Full night cycle (sample & send)
- 2.65mA @ 4.2V
- 25% of the time powered on
00:00:00 000.000
00:16:40 000.000
00:33:20 000.000
00:50:00 000.000
00:55:13 624.716
Δ51:23m
WINDOW
2.82mA average
310.01mA max
55:09m time
9.34C charge
SELECTION
ZOOM TO SELECTION | SELECT ALL | CLEAR
2.65mA average
250.91mA max
51:23m time
8.18C charge

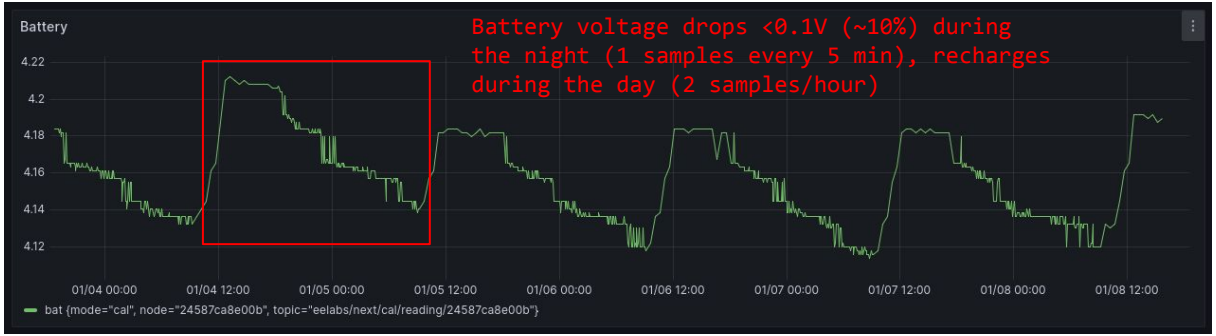

Lock Y-axis
Δ55:13m
400 mA
300 mA
200 mA
100 mA
0 nA
00:00:00 000.000
00:16:40 000.000
00:33:20 000.000
00:50:00 000.000
00:55:13 624.716
Δ51:23m
WINDOW
2.82mA average
310.01mA max
55:09m time
9.34C charge
SELECTION
ZOOM TO SELECTION | SELECT ALL | CLEAR
2.65mA average
250.91mA max
51:23m time
8.18C charge

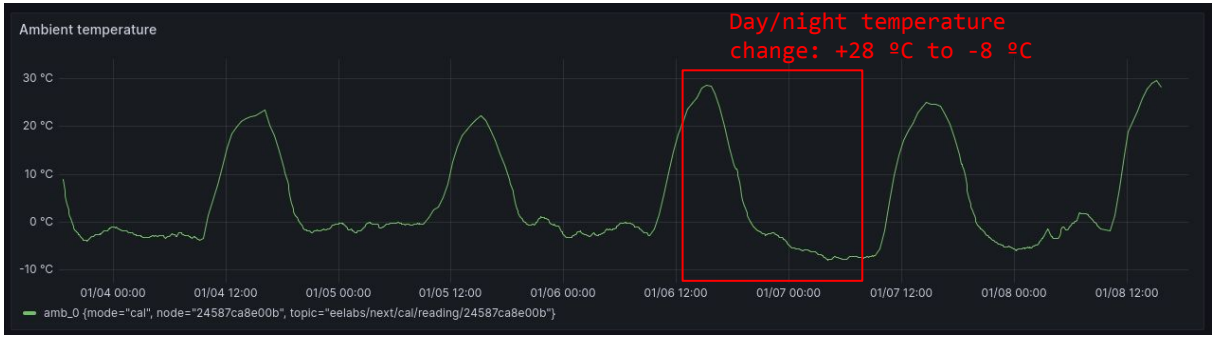

Lock Y-axis
1s
3s
10s
1min
10min
1h
LIVE VIEW
Δ1:25.7m
30 mA
25 mA
20 mA
15 mA
10 mA
5 mA
0 nA
Sampling cycle (~75 s)
00:06:08
015.489
00:06:40
000.000
00:07:00
000.000
00:07:33
809.439
Δ1:18.5m
WINDOW
8.54mA
average
29.90mA
max
1:25.7m
time
0.73C
charge
SELECTION
ZOOM TO SELECTION
SELECT ALL
CLEAR
9.34mA
average
29.90mA
max
1:18.5m
time
0.73C
charge

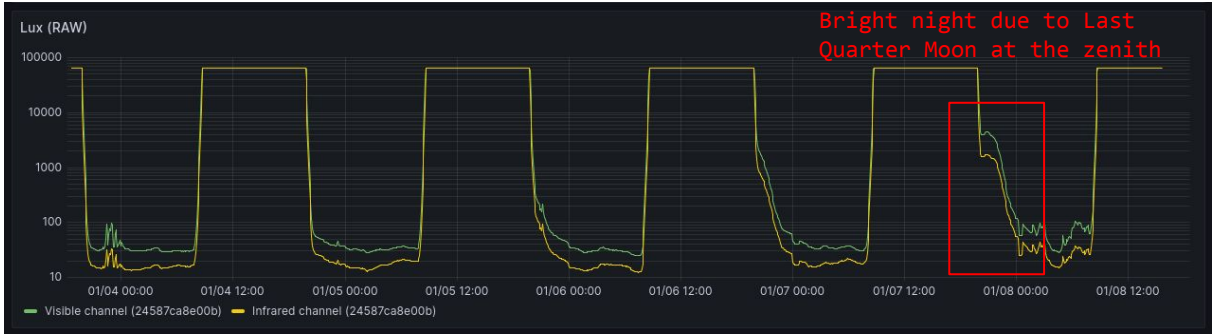

Lock Y-axis
Δ55:13m
400 mA
300 mA
200 mA
100 mA
0 nA
00:00:00 000.000
00:16:40 000.000
00:33:20 000.000
00:50:00 000.000
00:55:13 624.716
Δ51:23m
WINDOW
2.82mA average
310.01mA max
55:09m time
9.34C charge
SELECTION
ZOOM TO SELECTION
SELECT ALL
CLEAR
2.65mA average
250.91mA max
51:23m time
8.18C charge


Lock Y-axis
1s
3s
10s
1min
10min
1h
LIVE VIEW
Δ1:47.2m
300 mA
250 mA
200 mA
150 mA
100 mA
50 mA
0 nA
Sampling & connection cycle
00:50:58
647.827
00:51:40
000.000
00:52:45
890.274
Δ1:29.6m
WINDOW
13.20mA
average
250.91mA
max
1:47.2m
time
1.42C
charge
SELECTION
ZOOM TO SELECTION
SELECT ALL
CLEAR
15.81mA
average
250.91mA
max
1:29.6m
time
1.42C
charge

THE RESULTS







Teide Observatory (2,390 m)
4th-8th Jan 2025

Thank you!

✉️ x.crespo@scrobotics.es

🌐 https://gitlab.com/scrobotics/optical-makerspace/dark-sky-meter-fw

🌐 https://gitlab.com/scrobotics/optical-makerspace/dark-sky-meter-hw