扉開閉記録装置 K-Logger

Version 2.3

1. 概要

建築内の執務者の移動や室間換気の量を推定する上で、扉の開閉状態を把握したいという場面は多い。K-Logger は距離センサを応用することで扉の開閉状態を長期に記録するための計測器である。

基板面に設けられた距離センサにより、1 秒ごとに約 80 cm までの距離を計測し、micro SD カードに保存できる。単 3 電池で約 3 週間の連続測定ができる。重量は 37g(本体 7g; 電池 30g)と軽量のため、養生テープなどを使って扉の周囲の壁面などに容易に固定できる。

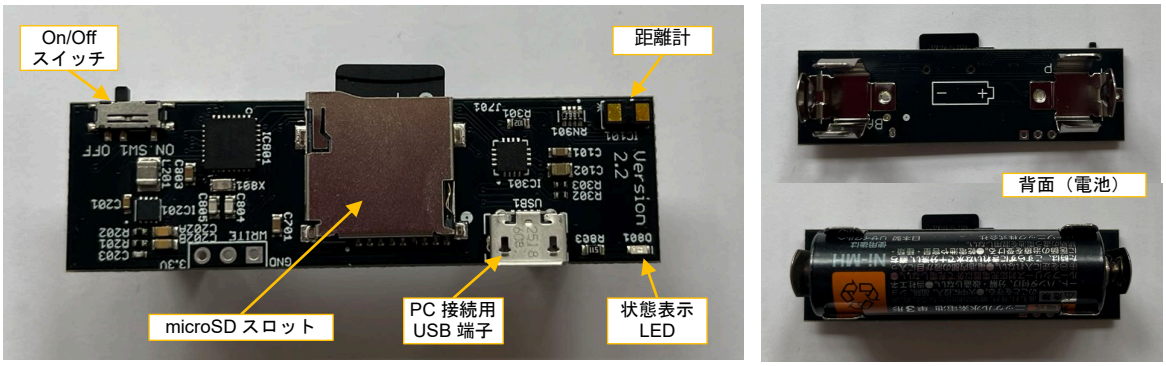


図 1.1　計測器の外観（左：表面、右：裏面）

2. 使い方

2.1 基礎

K-Logger の背面に電池を入れて表面のスイッチを On にすると、状態表示 LED が 1 秒間隔の点滅を始める。microSD をスロットに挿すと状態表示 LED が 10 秒に 1 回の点滅に変わる。これで、距離計から垂直方向の障害物までの距離が毎秒計測されて記録される。ただし、電力消費を削減するため、microSD には 1 分に 1 回の頻度でしか書き出されない。従って、少なくとも起動してから 1 分以上は経過しないとデータが書き出されないことに注意されたい。

スイッチを Off にして microSD を PC などで読み込むと「xxxx.csv」というファイル名が確認できる(図 1.1)。xxxx は数字 4 桁で、計測器を On にするか、1 日(86400 秒)が経過する度に数値が増えていく。

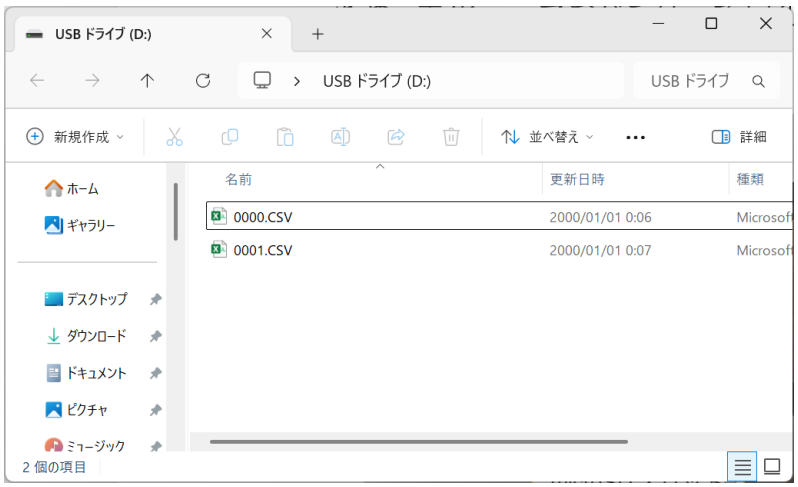
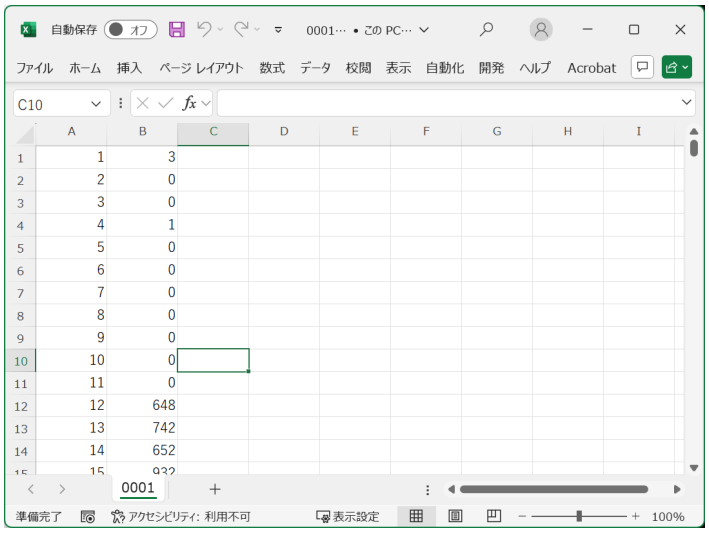


図 2.1　作成されたファイル

Excel などで CSV ファイルを開くと、1 列目に秒数、2 列目に距離[mm]が記録されている（図 2.1）。ここで、秒数は計測器を起動してからの経過秒数である。



	A	B
1	1	3
2	2	0
3	3	0
4	4	1
5	5	0
6	6	0
7	7	0
8	8	0
9	9	0
10	10	0
11	11	0
12	12	648
13	13	742
14	14	652

図 2.2　記録結果

2.2　日時設定とキャリブレーション（PC との接続）

1) ドライバのインストール

USB ケーブルを使って PC と接続することで、現在の日時の設定と距離のキャリブレーションができる。ただし、電池の消費を削減するため、PC と通信ができるのは起動後 60 秒間のみである。

本計測器は CP2102N を使って USB 経由で Windows と UART で通信する。このため、CP2102N のドライバをインストールする必要がある。このため、以下のサイトから「CP210x Universal Windows Driver」をダウンロードして解凍する。

https://www.silabs.com/software-and-tools/usb-to-uart-bridge-vcp-drivers

Windows で「デバイスマネージャー」を起動した後、計測器の電源を入れて USB ケーブルで PC と接続する。デバイスマネージャに「CP2102N USB to UART Bridge Controller」が表示されるが、ドライバがインストールされていないため、図 2.3 に示すようにエクスクラメーションマークが表示される。

この項目を選び、右クリックで「ドライバの更新」を選択し、「コンピュータを参照してドライバーを検索」をクリックする。先ほど解凍した CP210x のドライバのフォルダを選択するとドライバがインストールされる。正常にインストールされるとデバイスマネージャのポート一覧に登録される（図 2.4）。

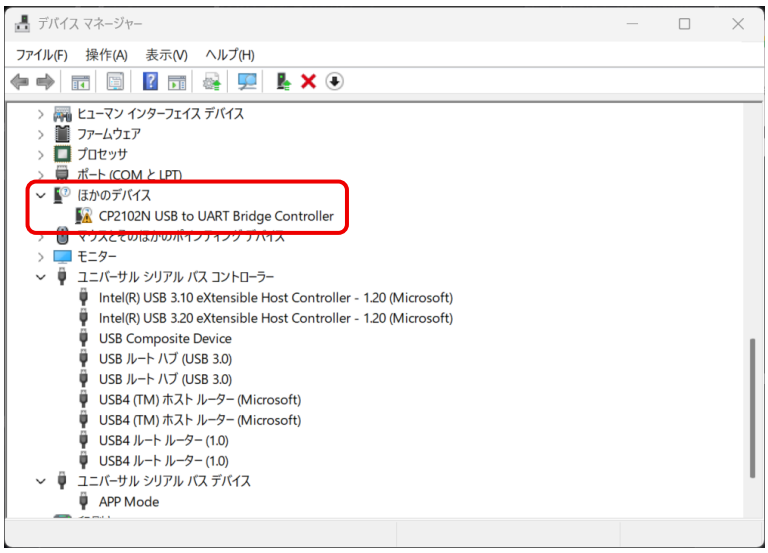


図 2.3　ドライバのインストール前の CP2102N の認識

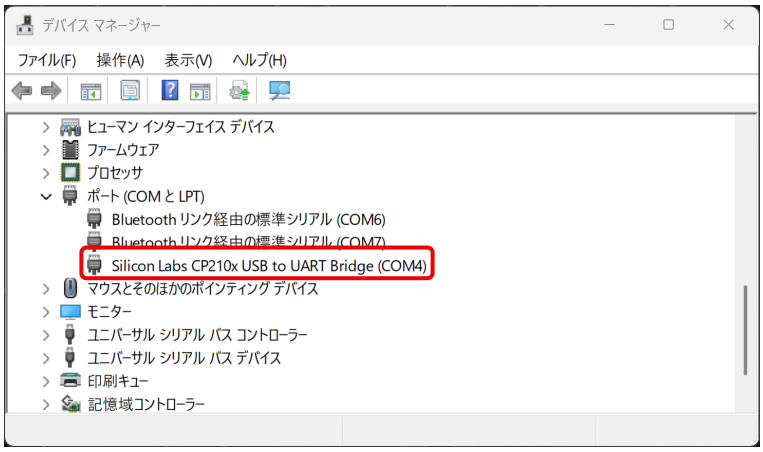


図 2.4　ドライバのインストール後の CP2102N の認識

UART 通信で特定のコマンドを送ることで日時設定とキャリブレーションができる。Windows PowerShell を使ってこれを自動化するためのバッチファイルを提供している。これらのファイル(「K-Logger-Tool.ps1」「SetTime.bat」「Calibrate.bat」「Measure.bat」)は同一階層に置いて使う。

2) 日時の設定

計測器を起動して Windows につなぎ、「SetTime.bat」をダブルクリックすると Windows 側の現在時刻情報が計測器に送信される（図 2.5）。

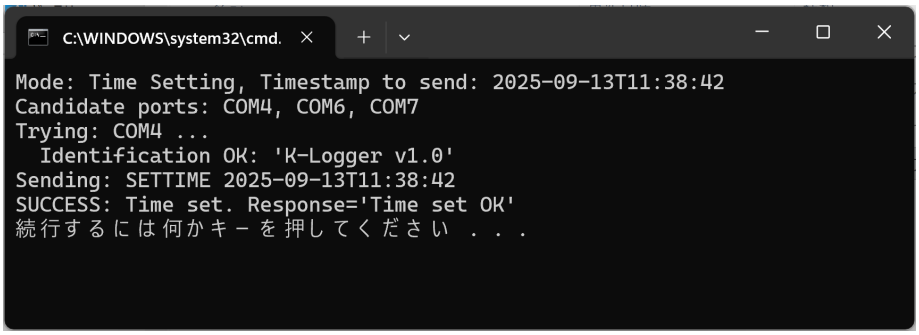


図 2.5　日時の設定

現在日時が計測器に反映されると、記録されるファイル名称は「yyyyMMdd.csv」に変わる（図 2.6）。

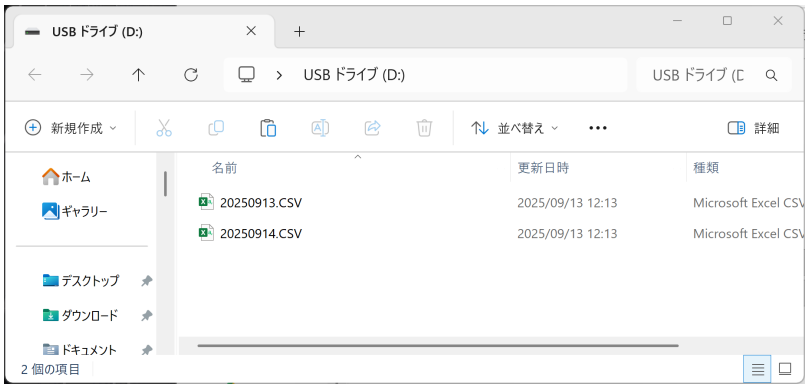


図 2.6　作成されたデータ

また、記録される CSV データの 1 列目は「HH:mm:ss」形式の時刻データに変わる（図 2.7）。

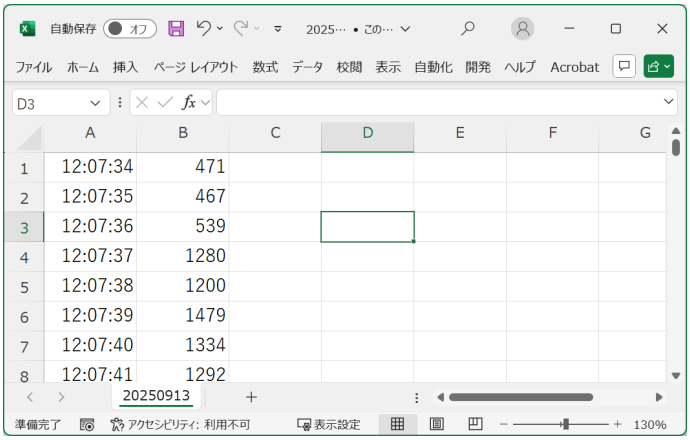


図 2.7　記録結果

3) キャリブレーション

測定距離の精度を上げるため、キャリブレーションコマンドが用意されている。計測器から 14 cm の位置に反射率 17%のグレーの物体を置く(図 2.8)。カメラの露出調整のために反射率 18%の標準グレーカードが広く流通しているため、これを使えば良い。

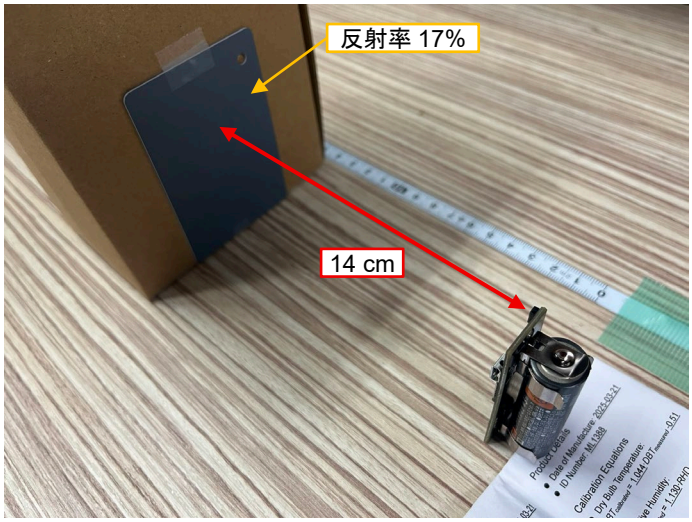


図 2.8　キャリブレーション準備

計測器を起動して Windows につなぎ、「Calibrate.bat」をダブルクリックすると 7 秒ほどでキャリブレーションが終わり、補正距離が出力される(図 2.9)。キャリブレーションの結果は計測器内部に保存されるため、電源の On/Off によって初期化されることはない。

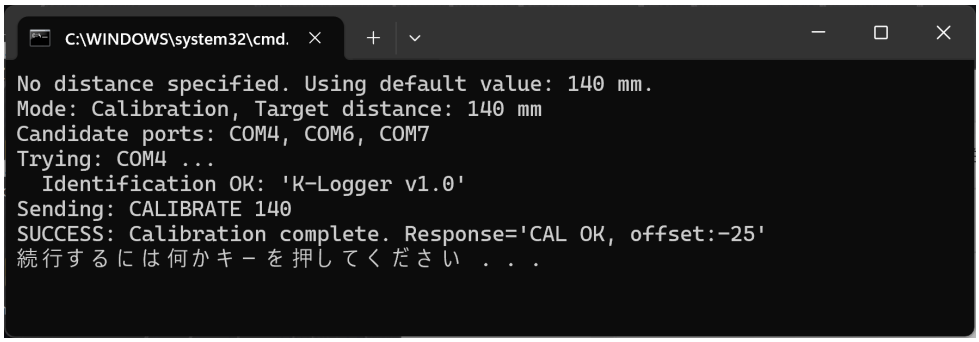


図 2.9　キャリブレーション

4) リアルタイム計測

計測器を起動して Windows につなぎ、「Measure.bat」をダブルクリックすると現在の計測値がリアルタイムで Windows に表示される。これはテスト用の機能で、計測値は 0.1 秒間隔で素早く更新されるかわりに、CSV には書き出されない。

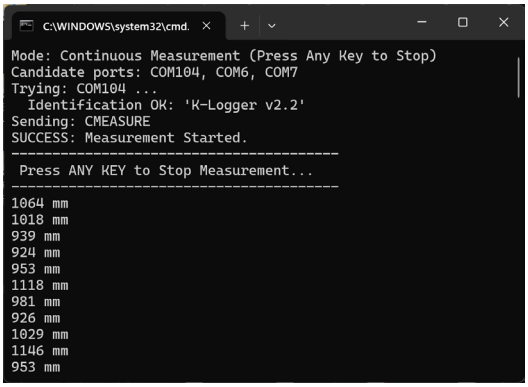


図 2.10　リアルタイム計測

2.3　実例

1) 設置方法

図 2.11～図 2.13 に設置方法を示す。図 2.11 に示すように通常の利用を妨げないように扉の上部に取り付ける。軽量のため、養生テープで十分に固定できる（図 2.12）。図 2.13 は扉を開放した写真である。計測部が扉を向くように取り付ける。図 2.14 に示すように扉の根本側に取り付ければ、扉の開度に応じて距離が異なることを利用して開度を推定することができる。

図 2.11　扉への設置

図 2.12　拡大

図 2.13　扉開放時

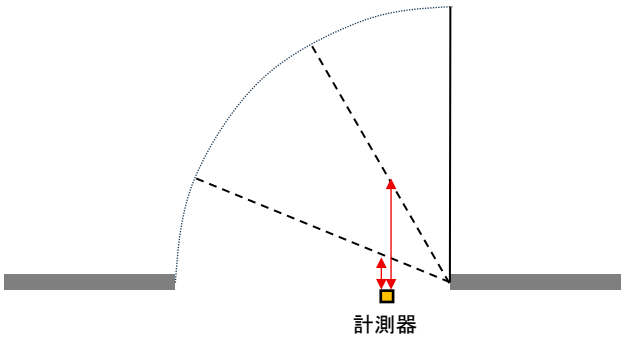


図 2.14　扉の開度と測定距離

2) 扉開度の推定法

図 2.15 に扉の開度の推定法を示す。幅L [mm]の扉の根本からw [mm]の位置に計測器を取り付ける。扉の開度がα [度]のときの計測値をx_α [mm]とする。ただし、通常は全閉でも隙間はあるため、α=0 であっても計測値は 0 mm とはならない。全閉の時の計測値をx_0[mm]とする。

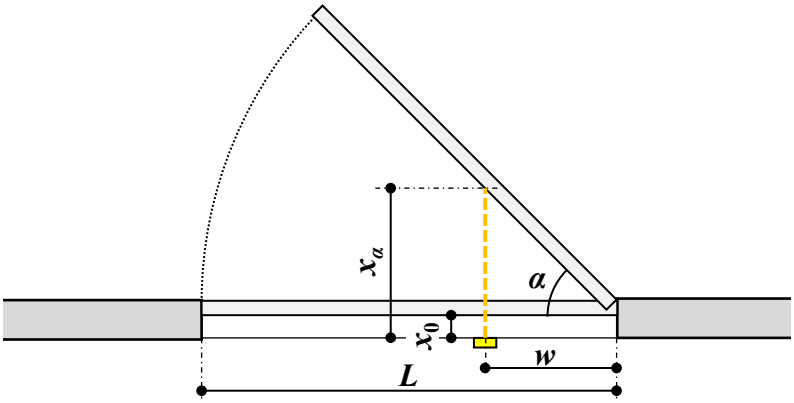


図 2.15　扉の開度の推定法

開度αまで計測したい場合には、計測値の取り付け位置は$w < L\cos\alpha$を満たさなければならない。逆にwの位置に設置した場合には開度$\alpha = \arccos(w/L)$まで計測できる。

計測値がx_αのとき、開度は$\alpha = \arctan\{(x_\alpha - x_0)/w\}$となる。

幅 835 mm の扉の根本から 260 mm の位置に計測器を取り付け、30 秒ごとに 15 度ずつ扉の開度を広げていった。図 2.16 に結果を示す。

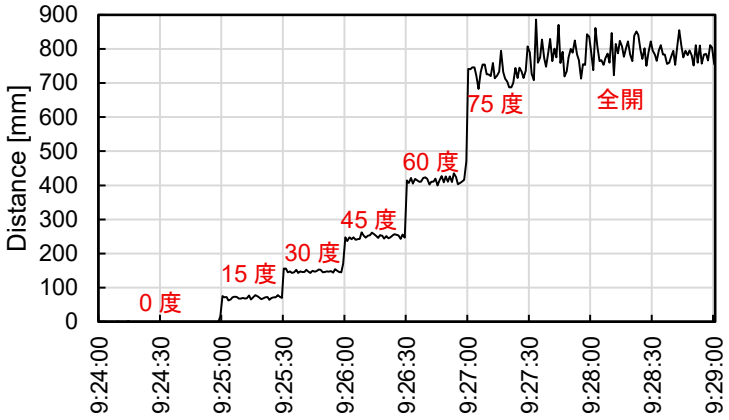


図 2.16　扉開度と距離計測値の関係

扉の開度が広がるにつれて距離計の計測値も大きくなる。15、30、45、60、75 度のときの平均距離はそれぞれ 71.1、148.4、251.9、413.9、731.5 mm となった。先の式に代入すると開度の予測値は 15.3、29.7、44.1、57.9、70.4 度となる。なお、本例では 0 度のときに計測値が 0 となったため$x_0 = 0$とした。

3. 技術情報

3.1 距離計測の精度

距離計測の精度を確認するため、段階的に距離を離していき、その時の計測値を記録した。ただし、反射率 18%のグレーカードを 14cm 離した状態でのキャリブレーションは実施済みの計測器である。

結果を図 3.1 に示す。80cm 程度が計測の限界で、それ以上の距離を取るとほぼ一定になる。80cm 未満の距離であれば、誤差は最大で 3cm 程度だった。

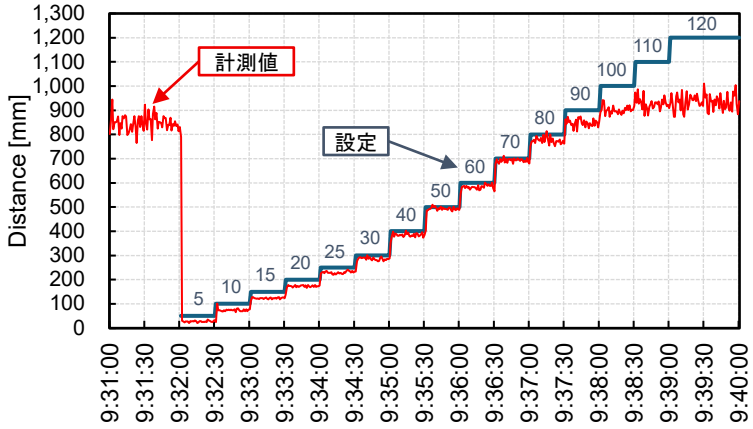


図 3.1　距離計測の精度

3.2 電池の消費特性

計測器が起動してから安定状態に入るまでの電力消費の特性を図 3.2 に示す。単 3 電池での動作を前提とするため、1.2 V の安定化電源に接続した。起動して 1 分間は UART 通信に対応するため、7mA 程度の電流が流れる。その後はマイコンが深いスリープモードに入り、電流が減少する。microSD への書き出しのために 1 分に 1 回、やや大きい電流が流れる。1 分あたりの平均電流は 2.7 mA 程度である。Eneloop（単 3）の容量は 2,500 mAh のため、2500÷2.7÷24=38 日程度の連続稼働が期待できる。

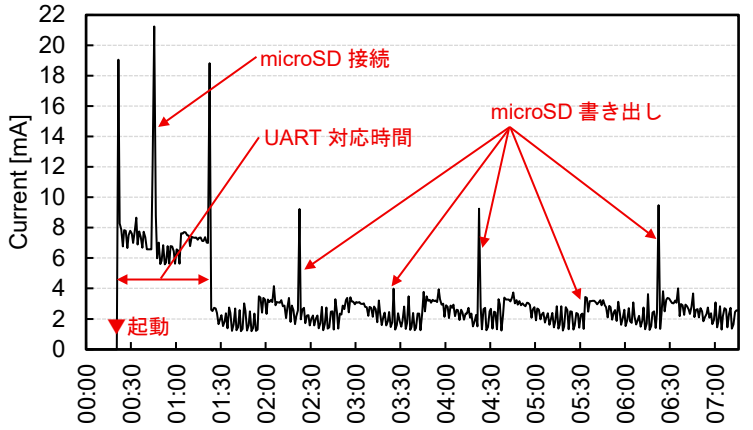


図 3.2　電流値

単 4 電池を使った加速試験では、9 月 13 日 17:00 から計測を始めて 9 月 29 日 21:14 まで 388.2 時間（16.2 日）の連続計測ができた。試験に使った Eneloop Pro 単 4 の容量は 930 mAh で、単 3 は 2500 mAh である。従って、電池の消費が比例的だと仮定すれば、単 3 であれば、1043.6 時間（43.5 日）の連続計測ができることになる。

3.3 時刻の誤差

2025 年 9 月 15 日 9:32:18 に時刻を合わせて計測を始め、9 月 18 日 12:52 に終了したとき、誤差は 1 秒未満であった。

3.4 PC 通信用バッチファイル

Windows と計測器は USB を介して UART（ボーレート 9600、パリティなし）で通信する。時刻設定、キャリブレーション、テスト用リアルタイム計測の 3 つのコマンドがあり、以下の仕様である。

時刻設定： SETTIME yyyy-MM-ddTHH:mm:ss

キャリブレーション： CALIBRATE

リアルタイム計測： CMEASURE

Windows からのコマンド送信を自動化するためのバッチファイルを以下に示す。時刻設定の場合には「SetTime.bat」、キャリブレーションの場合には「Calibrate.bat」、リアルタイム計測の場合には「Measure.bat」をダブルクリックすれば良い。

K-Logger-Tool.ps1

```
param(
  [int]$Baud = 9600,
  [string]$IdentifyCmd = "ID?",
  [string]$IdentifyExpect = "K-Logger",
  [string]$TimeSetCmd = "SETTIME",
  [string]$TimeSetAck = "Time set OK",
  [string]$CalibrateCmd = "CALIBRATE",
  [string]$CalibrateAck = "CAL OK",
  [string]$MeasureCmd = "CMEASURE",
  [string]$MeasureAck = "Start continuous measuring",
  [string]$StopCmd = "STOP",
  [string]$StopAck = "Stop continuous measuring",
  [switch]$ContinuousMode, # 連続計測モード用スイッチ
  [int]$ReadTimeoutMs = 7000,
  [int]$WriteDelayMs = 50,
  [int]$CalibrateDistance = -1
)

# ---- Get candidate COM ports ----
function Get-CandidatePorts {
  return ([System.IO.Ports.SerialPort]::GetPortNames() | Sort-Object)
}

# ---- Open COM port ----
function New-OpenPort([string]$name, [int]$baud) {
  $sp = New-Object System.IO.Ports.SerialPort $name,$baud,'None',8,'One'
  $sp.ReadTimeout  = $ReadTimeoutMs
  $sp.WriteTimeout = 500
  $sp.NewLine      = "`r`n"
  $sp.Encoding     = [System.Text.Encoding]::ASCII
  try { $sp.Open() } catch { return $null }
  return $sp
}

# ---- Send one line ----
function Send-Line([System.IO.Ports.SerialPort]$sp, [string]$line) {
```

```
  try {
    $sp.DiscardInBuffer()
    $sp.WriteLine($line)
    Start-Sleep -Milliseconds $WriteDelayMs
    return $true
  } catch { return $false }
}

# ---- Read one line safely ----
function Read-LineSafe([System.IO.Ports.SerialPort]$sp) {
  try { return $sp.ReadLine().Trim() } catch { return $null }
}


# ---- Main ----
$timestamp = Get-Date -Format 'yyyy-MM-ddTHH:mm:ss'

if ($ContinuousMode) {
    Write-Host "Mode: Continuous Measurement (Press Any Key to Stop)"
} elseif ($CalibrateDistance -ge 0) {
    Write-Host "Mode: Calibration, Target distance: $CalibrateDistance mm"
} else {
    Write-Host "Mode: Time Setting, Timestamp to send: $timestamp"
}

$candidates = Get-CandidatePorts
if ($candidates.Count -eq 0) {
  Write-Host "ERROR: No COM port found. Check USB connection and driver."
  exit 1
}

Write-Host "Candidate ports: $($candidates -join ', ')"
$target = $null

foreach ($p in $candidates) {
  Write-Host "Trying: $p ..."
  $sp = New-OpenPort -name $p -baud $Baud
  if (-not $sp) { Write-Host "  Could not open."; continue }

  if (-not (Send-Line $sp $IdentifyCmd)) { $sp.Close(); continue }

  $resp = Read-LineSafe $sp
  if ($resp -and $resp.Contains($IdentifyExpect)) {
    Write-Host "  Identification OK: '$resp'"
    $target = $sp
    break
  } else {
    Write-Host "  Identification failed: response='$resp'"
    $sp.Close()
  }
}

if (-not $target) {
  Write-Host "ERROR: Target device not found."
  exit 2
}

# --- Command Logic ---
if ($ContinuousMode) {
    # --- 連続計測モード ---

    # 1. 計測開始コマンド送信
    Write-Host "Sending: $MeasureCmd"
    if (-not (Send-Line $target $MeasureCmd)) {
        Write-Host "ERROR: Send failed"; $target.Close(); exit 3
    }

    # 2. 開始 ACK 確認
    $resp = Read-LineSafe $target
```

```
# main.cpp の出力に合わせて判定
if ($resp -and $resp.Contains("Start continuous measuring")) {
    Write-Host "SUCCESS: Measurement Started."
    Write-Host "---------------------------------------"
    Write-Host " Press ANY KEY to Stop Measurement... "
    Write-Host "---------------------------------------"
} else {
    Write-Host "WARNING: Unexpected start response. Response='$resp'"
    # ACK がおかしくてもデータが来るかもしれないので続行してみる
}

# 3. ループ処理（キー入力監視）
# ユーザー入力を受け付けるため、読み込みタイムアウトを一時的に短くする
$originalTimeout = $target.ReadTimeout
$target.ReadTimeout = 500 # 0.5 秒ごとにキー入力をチェック

try {
    while ($target.IsOpen) {
        # --- キー入力チェック ---
        if ([Console]::KeyAvailable) {
            # キーバッファを読み捨ててループを抜ける
            $null = [Console]::ReadKey($true)
            Write-Host "`r`nStopping requested by user..."
            break
        }

        # --- データ受信 ---
        try {
            $line = $target.ReadLine()
            if ($line) {
                $trimmed = $line.Trim()
                # "MES "で始まるデータなら整形して表示
                if ($trimmed.StartsWith("MES ")) {
                    # "MES " (4 文字) を削除して " mm" を付ける
                    $val = $trimmed.Substring(4)
                    Write-Host "$val mm"
                } else {
                    # それ以外（エラーや ACK など）はそのまま表示
                    Write-Host $trimmed
                }
            }
        } catch [System.TimeoutException] {
            # タイムアウトは無視してループ継続（キー入力チェックのため）
        } catch {
            Write-Host "Read Error: $_"
            break
        }
    }
} finally {
    # タイムアウト設定を戻す
    $target.ReadTimeout = $originalTimeout
}

# 4. 停止コマンド送信
Write-Host "Sending: $StopCmd"
# バッファに残っているデータを捨ててから送信
$target.DiscardInBuffer()

if (-not (Send-Line $target $StopCmd)) {
    Write-Host "ERROR: Stop Send failed"; $target.Close(); exit 3
}

# 5. 停止 ACK 確認
$resp = Read-LineSafe $target
$target.Close()

if ($resp -and $resp.Contains("Stop continuous measuring")) {
    Write-Host "SUCCESS: Measurement Stopped cleanly."
    exit 0
```

```
    } else {
        Write-Host "WARNING: Stopped but unexpected response. Response='$resp'"
        exit 0
    }

} elseif ($CalibrateDistance -ge 0) {
    # Calibration command
    $cmd = "$CalibrateCmd $CalibrateDistance"
    Write-Host "Sending: $cmd"
    if (-not (Send-Line $target $cmd)) {
        Write-Host "ERROR: Send failed"; $target.Close(); exit 3
    }
    $resp = Read-LineSafe $target
    $target.Close()

    if ($resp -and $resp.Contains($CalibrateAck)) {
        Write-Host "SUCCESS: Calibration complete. Response='$resp'"
        exit 0
    } else {
        Write-Host "ERROR: Calibration failed. Response='$resp'"
        exit 4
    }
} else {
    # Time setting command
    $cmd = "$TimeSetCmd $timestamp"
    Write-Host "Sending: $cmd"
    if (-not (Send-Line $target $cmd)) {
      Write-Host "ERROR: Send failed"; $target.Close(); exit 3
    }

    $resp = Read-LineSafe $target
    $target.Close()

    if ($resp -and $resp.Contains($TimeSetAck)) {
      Write-Host "SUCCESS: Time set. Response='$resp'"
      exit 0
    } else {
      Write-Host "WARNING: No valid response. Response='$resp'"
      exit 4
    }
}
```

SetTime.bat

```
@echo off
rem K-Logger に現在時刻を設定する。

powershell -ExecutionPolicy Bypass -File "%~dp0K-Logger-Tool.ps1"

pause
```

Calibrate.bat

```
@echo off
rem K-Logger の距離センサーをキャリブレーションする。
rem 引数なしで実行すると、メーカー推奨の 140mm で実行される。
rem 使い方: Calibrate.bat [距離[mm]]

set "DISTANCE=%1"

rem 引数が指定されていないかチェック
if "%DISTANCE%"=="" (
    echo No distance specified. Using default value: 140 mm.
    set "DISTANCE=140"
) else (
    echo Using specified distance: %DISTANCE% mm.
)

powershell -ExecutionPolicy Bypass -File "%~dp0K-Logger-Tool.ps1" -CalibrateDistance %DISTANCE%
```

```
pause
```

Measure.bat

```
@echo off

rem K-Logger で連続計測を開始し、データを表示する

powershell -ExecutionPolicy Bypass -File "%~dp0K-Logger-Tool.ps1" -ContinuousMode

pause
```