Challenger RP2040 UWB (DWM3000)

Progress Documentation

Siddharth Patel

HTW Berlin

Document Summary

This document tracks the steps taken to bring up the iLabs Challenger RP2040 UWB board, program it using the Arduino toolchain, run initial UWB examples, and explore how distance data can be integrated into a larger robotics system (e.g. Raspberry Pi 5).

Board	iLabs Challenger RP2040 UWB (DWM3000)
UWB Chip	Qorvo DW3000 (DWM3000 module)
Main MCU	Raspberry Pi RP2040
Toolchain	Arduino IDE (RP2040 core)
Date	December 5, 2025

Contents

List of Figures

List of Tables

1 Programming Options for the Challenger RP2040 UWB

1.1 RP2040 + DWM3000: Board Overview

The *Challenger RP2040 UWB* from iLabs combines

$$Challenger RP2040 UWB \;=\; RP2040(MCU) \;+\; DWM3000(UWB).$$

The RP2040 defines how we can program the board, while the DWM3000 defines which UWB libraries we can actually use.

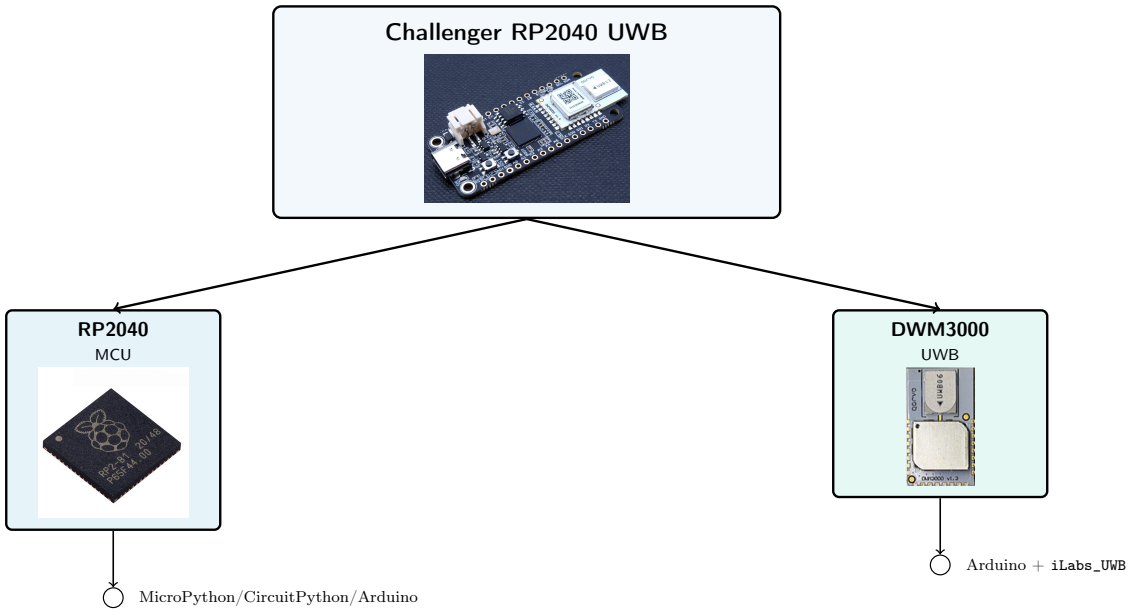

Figure 1: Conceptual view of the Challenger RP2040 UWB: RP2040 MCU, DWM3000 UWB module and their firmware options.

1.2 Firmware Options on the RP2040

Because the RP2040 is the same MCU used on the Raspberry Pi Pico, it can in principle be programmed with:

- **MicroPython**[1]
- **CircuitPython**[2]
- **Arduino (C++ with Arduino IDE)**[3]

1.3 Library Support for the DWM3000

For the UWB part, the situation is much more restricted:

- The iLabs product page notes that, although the board is compatible with CircuitPython, *"to date there does not exist a support library in Python for the DWM3000 module."*[4]

[1] https://micropython.org/download/?mcu=rp2040
[2] https://circuitpython.org/board/raspberry_pi_pico/
[3] https://ilabs.se/getting-your-challenger-rp2040-board-up-and-running-with-the-arduino-ide/
[4] https://ilabs.se/product/challenger-rp2040-uwb/

- iLabs provides an official **Arduino library** and examples for the DWM3000 via the `iLabs_UWB` repository.[5]

1.4 Conclusion for This Project

Because only the Arduino ecosystem currently offers a maintained UWB library for the DWM3000, all experiments in this document are based on:

$$RP2040 firmware = \textbf{Arduino language} + \textbf{Arduino IDE} + \texttt{iLabs\_UWB}\ \textbf{library}.$$

MicroPython and CircuitPython remain theoretical options for the RP2040 itself, but are not used here due to the missing DWM3000 support.

[5] https://github.com/PontusO/iLabs_UWB

2 Arduino-Based Workflow for the Challenger RP2040 UWB

2.1 Following the iLabs Getting-Started Guide

The official starting point for programming the Challenger RP2040 UWB is the iLabs Arduino guide.[6] It explains how to install the iLabs board package and select the correct board in the Arduino IDE.

Basic setup (summary)

1. Install the iLabs RP2040 board package in the Arduino IDE as described in the guide.
2. In `Tools` → `Board`, select `iLabs Challenger 2040 UWB`.
3. Verify that a serial port for the board (e.g. `/dev/cu.usbmodemXXXX`) is visible.

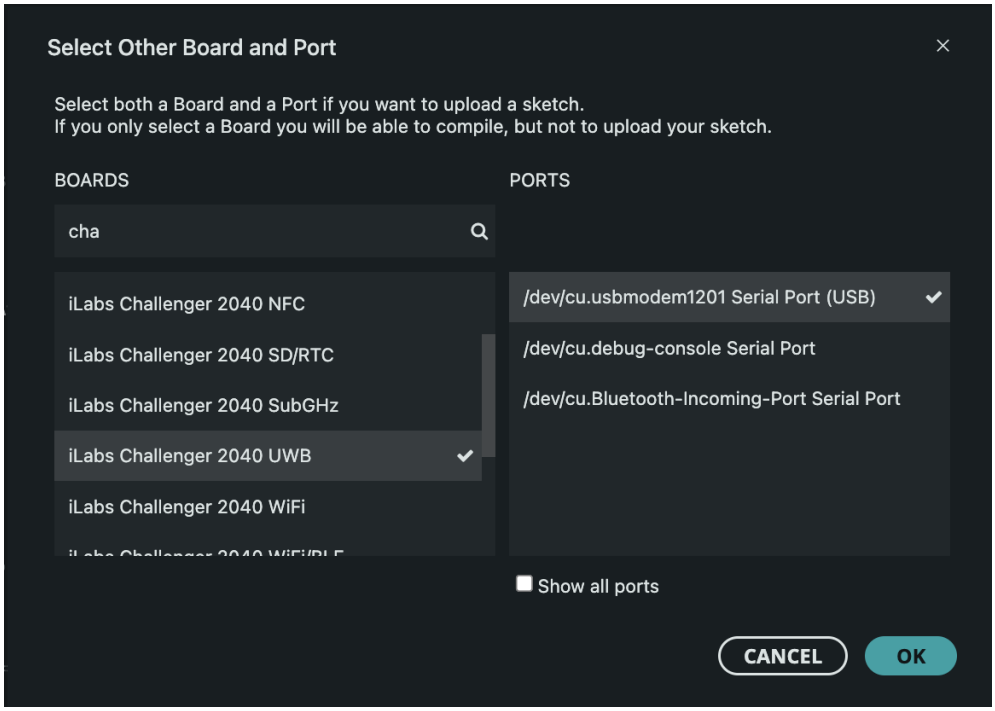

Figure 2: Selecting the `iLabs Challenger 2040 UWB` board and USB serial port in the Arduino IDE.

2.2 Bootloader Issue: "No Drive Detected"

When uploading directly from the Arduino IDE, the RP2040 bootloader is supposed to mount as a USB drive. On macOS this sometimes fails and the IDE reports:

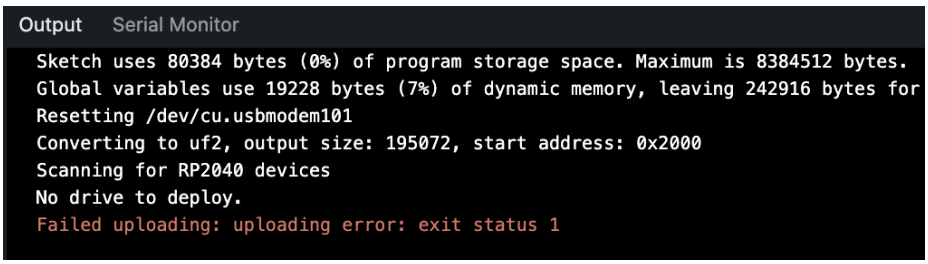

Figure 3: Typical Arduino upload error when the RP2040 UF2 drive is not detected.

To work around this, sketches are uploaded via a `.uf2` file and manual drag&drop.

[6] https://ilabs.se/getting-your-challenger-rp2040-board-up-and-running-with-the-arduino-ide/

2.3 Programming via UF2 Files

The UF2 method separates two things:

1. the Arduino IDE *compiles* a `.uf2` file, and
2. the Mac *copies* that file to the RP2040 boot drive.

2.3.1 One-time setup in the Arduino IDE

Step 1: IDE configuration

1. Open your sketch in the Arduino IDE.
2. In `Tools` set:
   - `Board` → `iLabs Challenger 2040 UWB`,
   - `Upload Method` → `Default (UF2)`,
   - `USB Stack` → `Pico SDK`.
3. Choose `Sketch` → `Export compiled Binary`.
4. Open `Sketch` → `Show Sketch Folder` and locate the generated `.uf2` file (e.g. `my_sketch.ino.uf2`).

2.3.2 Put the board into BOOTSEL mode

Step 2: Enter BOOTSEL mode

1. Unplug the USB cable from the Challenger RP2040 UWB.
2. Hold the **BOOT** button and plug the USB cable back in.
3. Release **BOOT** when a new USB drive named `RPI-RP2` appears on the Mac.

2.3.3 Upload the program via UF2

Step 3: Copy UF2 file

1. Drag the `.uf2` file from the sketch folder onto the `RPI-RP2` drive in Finder.
2. Wait until `RPI-RP2` automatically ejects.
3. Unplug and reconnect the board in normal mode. The new sketch is now running.

2.4 FreeRTOS Configuration Error

Some UWB example sketches use FreeRTOS on the RP2040. If FreeRTOS is not enabled, compilation fails with an error similar to:

```
    4 | #error "#define __FREERTOS 1 to use FreeRTOS in your application"
      |  ^~~~~
exit status 1

Compilation error: exit status 1
```

Figure 4: Compilation error when FreeRTOS is not enabled for the Challenger RP2040 UWB.

2.4.1 Enable FreeRTOS for the Challenger RP2040 UWB

Fix: enable FreeRTOS

1. Open the example sketch (e.g. `ex_01a_simple_tx.ino`) in the Arduino IDE.
2. In `Tools` set:
   - `Board` → `iLabs Challenger 2040 UWB`,
   - `Upload Method` → `Default (UF2)`,
   - `USB Stack` → `Pico SDK`,
   - `Operating System` → **FreeRTOS SMP**.
3. Compile again with `Sketch` → `Verify/Compile`.
4. If compilation succeeds, upload the sketch via the UF2 workflow described above.

3 Which Data Can We Get from the DWM3000?

3.1 Raw data directly from the DWM3000

When the RP2040 talks to the DWM3000 over SPI, it can read several types of *raw* information from the chip registers.[7]

Table 1: Typical raw data available from the DWM3000

Raw data type (examples)
RX and TX timestamps in the internal UWB clock
Received UWB frame bytes (header fields and payload)
Channel impulse response (CIR) samples
RX quality and diagnostic values (power, noise, CRC status, . . .)
Status and interrupt flags (RXOK, timeout, errors, . . .)

These low-level values are the building blocks. From them we can compute things like time-of-flight, distance or more advanced metrics.

3.2 Derived data on RP2040 using the UWB library

With the iLabs UWB library running on the RP2040, these raw values are turned into more convenient, *processed* data structures:

Table 2: Examples of derived data on the RP2040

Processed data (examples)
Distance between two nodes (single range result in m or mm)
Simple range quality or “RSSI” value per distance measurement
Logged packets that combine payload, timestamps and diagnostics
2D or 3D position estimates of a tag based on multiple ranges
Debug recordings of CIR, timestamps and error counters

[7] Full details are in the DWM3000 datasheet: https://www.qorvo.com/products/p/DWM3000#documents

4 Retrieving Raw UWB Data on the RP2040

4.1 How to run the raw-data demo

Step-by-step setup

1. **Board A (RX)**: Flash `rx_raw_data_demo.ino` to the first Challenger RP2040 UWB (using the Arduino + UF2 workflow from Section 2).
2. Open the Arduino **Serial Monitor** for Board A with the baudrate used in the sketch (e.g. 115200 Bd).
3. **Board B (TX)**: Flash a TX example sketch, e.g. `ex_01a_simple_tx.ino` from the iLabs examples.[a]
4. Power both boards. Board B periodically sends UWB frames; Board A prints one raw dump for every received frame.

[a] https://github.com/PontusO/iLabs_UWB/blob/main/examples/ex_01a_simple_tx/ex_01a_simple_tx.ino

4.2 Example serial output

The TX board simply reports that frames are being sent, while the RX board prints a full raw dump for each frame (status, frame bytes, RX timestamp, CIR, power values, ...).

```
20:00:02.216 -> TX Frame Sent
20:00:02.742 -> TX Frame Sent
20:00:03.305 -> TX Frame Sent
20:00:03.831 -> TX Frame Sent
20:00:04.359 -> TX Frame Sent
20:00:04.887 -> TX Frame Sent
20:00:05.445 -> TX Frame Sent
20:00:05.976 -> TX Frame Sent
20:00:06.504 -> TX Frame Sent
20:00:07.032 -> TX Frame Sent
20:00:07.555 -> TX Frame Sent
20:00:08.118 -> TX Frame Sent
20:00:08.649 -> TX Frame Sent
```

Figure 5: * TX board: simple "TX Frame Sent" log from `ex_01a_simple_tx.ino`.

```
20:00:54.562 -> === RX FRAME ===
20:00:54.562 -> SYS_STATUS (low 32b): 0x1806F07
20:00:54.562 -> Frame length: 12
20:00:54.562 -> Frame bytes: C5 3B 44 45 43 41 57 41 56 45 05 02
20:00:54.562 -> RX timestamp (ticks): 2006772321
20:00:54.562 -> RX timestamp (full 40b, hex): 0x779CEA61
20:00:54.562 -> CIR first 128 bytes:
20:00:54.595 -> AD 0A 00 00 17 00 00 F2 FF FF E7 FF FF 12 00 00 F0 FF FF 48 00
20:00:54.595 -> ipatovPeak:   1572883075
20:00:54.595 -> ipatovPower:  46
20:00:54.595 -> ipatovFpIndex:47146
20:00:54.595 -> stsPeak:      0
20:00:54.595 -> stsPower:     0
20:00:54.595 -> stsFpIndex:   0
20:00:54.595 -> xtalOffset:   56
20:00:54.595 -> === END FRAME ===
```

Figure 6: * RX board: raw frame dump from `rx_raw_data_demo.ino` (timestamps, CIR, diagnostics).

Figure 7: Serial output of the TX example (left) and the raw-data receiver demo (right).

4.3 Sketch: `rx_raw_data_demo.ino`

The sketch `rx_raw_data_demo.ino` configures the Challenger RP2040 UWB as a receiver and prints the raw information from every incoming UWB frame (timestamps, frame bytes, CIR, diagnostics, ...) to the serial console.

```
#include "dw3000.h"

#define APP_NAME "RX RAW DATA DEMO"

// --- Basic UWB config (same as examples) ---
static dwt_config_t config = {
    5,               // Channel
    DWT_PLEN_128,    // Preamble length
    DWT_PAC8,        // PAC size
    9,               // TX preamble code
    9,               // RX preamble code
    1,               // SFD type
    DWT_BR_6M8,      // Data rate
    DWT_PHRMODE_STD, // PHY header mode
    DWT_PHRRATE_STD, // PHY header rate
    (129 + 8 - 8),   // SFD timeout
    DWT_STS_MODE_OFF,
    DWT_STS_LEN_64,
    DWT_PDOA_M0
};

// Antenna delays (same as Qorvo examples)
#define TX_ANT_DLY 16385
#define RX_ANT_DLY 16385

// RX buffer
#define RX_BUF_LEN 127
static uint8_t rx_buffer[RX_BUF_LEN];

// CIR buffer: just grab first 128 bytes as a demo
#define CIR_BUF_LEN 128
static uint8_t cir_buf[CIR_BUF_LEN];

// Diagnostics struct from DW3000 API
static dwt_rxdiag_t rx_diag;

// For status flags
static uint32_t status_reg = 0;

// From library (TX power etc.)
extern dwt_txconfig_t txconfig_options;

// Print bytes as hex
static void print_bytes_hex(const uint8_t *buf, uint16_t len)
```

```
{
    for (uint16_t i = 0; i < len; i++) {
        if (buf[i] < 0x10) Serial.print('0');
        Serial.print(buf[i], HEX);
        Serial.print(' ');
    }
    Serial.println();
}

void setup()
{
    Serial.begin(115200);
    delay(1000);
    Serial.println();
    Serial.println(APP_NAME);

    // Start SPI + DW3000
    spiBegin();
    spiSelect();
    delay(200);

    while (!dwt_checkidlerc()) {
        Serial.println("IDLE FAILED");
    }

    dwt_softreset();
    delay(200);

    while (!dwt_checkidlerc()) {
        Serial.println("IDLE FAILED (after reset)");
    }

    if (dwt_initialise(DWT_DW_INIT) == DWT_ERROR) {
        Serial.println("INIT FAILED");
        while (1);
    }

    if (dwt_configure(&config)) {
        Serial.println("CONFIG FAILED");
        while (1);
    }

    dwt_configuretxrf(&txconfig_options);

    dwt_setrxantennadelay(RX_ANT_DLY);
    dwt_settxantennadelay(TX_ANT_DLY);

    // Enable LNA/PA
    dwt_setlnapamode(DWT_LNA_ENABLE | DWT_PA_ENABLE);
```

```
94
95      // Enable CIA diagnostics so dwt_readdiagnostics() gives full data
96      // (1 = log full diagnostic set, see DW3xxx API Guide)
97      dwt_configciadiag(1);
98
99      Serial.println("Ready, waiting for RX frames...");
100 }
101
102 void loop()
103 {
104     // --- Put DW3000 into RX mode immediately ---
105     dwt_rxenable(DWT_START_RX_IMMEDIATE);
106
107     // Wait for good frame or RX error
108     while (!((status_reg = dwt_read32bitreg(SYS_STATUS_ID)) &
109              (SYS_STATUS_RXFCG_BIT_MASK | SYS_STATUS_ALL_RX_ERR))) {
110         // busy wait
111     }
112
113     if (status_reg & SYS_STATUS_RXFCG_BIT_MASK) {
114         // ----- 1) STATUS / FLAGS -----
115         Serial.println();
116         Serial.println("=== RX FRAME ===");
117         Serial.print("SYS_STATUS (low 32b): 0x");
118         Serial.println(status_reg, HEX);
119
120         // Clear good RX flag
121         dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_RXFCG_BIT_MASK);
122
123         // ----- 2) FRAME BUFFER -----
124         uint32_t frame_len = dwt_read32bitreg(RX_FINFO_ID) & RXFLEN_MASK;
125         if (frame_len > RX_BUF_LEN) frame_len = RX_BUF_LEN;
126
127         dwt_readrxdata(rx_buffer, frame_len, 0);
128
129         Serial.print("Frame length: ");
130         Serial.println(frame_len);
131
132         Serial.print("Frame bytes: ");
133         print_bytes_hex(rx_buffer, frame_len);
134
135         // ----- 3) RX TIMESTAMP -----
136         uint64_t rx_ts = get_rx_timestamp_u64();
137         Serial.print("RX timestamp (ticks): ");
138         Serial.println((unsigned long)(rx_ts & 0xFFFFFFFFUL));  // low 32 bits
139         Serial.print("RX timestamp (full 40b, hex): 0x");
140         Serial.println((unsigned long)(rx_ts & 0xFFFFFFFFUL), HEX);
141
142         // ----- 4) CIR ACCUMULATOR (first bytes) -----
```

```
143         dwt_readaccdata(cir_buf, CIR_BUF_LEN, 0); // read from accumulator mem
144 
145         Serial.print("CIR first ");
146         Serial.print(CIR_BUF_LEN);
147         Serial.println(" bytes:");
148         print_bytes_hex(cir_buf, CIR_BUF_LEN);
149 
150         // ----- 5) RX DIAGNOSTICS / QUALITY -----
151         dwt_readdiagnostics(&rx_diag);
152 
153         Serial.print("ipatovPeak:   ");
154         Serial.println(rx_diag.ipatovPeak);
155         Serial.print("ipatovPower:  ");
156         Serial.println(rx_diag.ipatovPower);
157         Serial.print("ipatovFpIndex:");
158         Serial.println(rx_diag.ipatovFpIndex);
159         Serial.print("stsPeak:      ");
160         Serial.println(rx_diag.stsPeak);
161         Serial.print("stsPower:     ");
162         Serial.println(rx_diag.stsPower);
163         Serial.print("stsFpIndex:   ");
164         Serial.println(rx_diag.stsFpIndex);
165         Serial.print("xtalOffset:   ");
166         Serial.println(rx_diag.xtalOffset);
167 
168         Serial.println("=== END FRAME ===");
169 
170     } else {
171         // RX error → clear flags and try again
172         Serial.println("RX ERROR");
173         dwt_write32bitreg(SYS_STATUS_ID, SYS_STATUS_ALL_RX_ERR);
174     }
175 
176     delay(100); // small pause between RX cycles
177 }
```

5 Example: Distance Measurement Between Two Nodes

For a first end–to–end distance test we used the single–sided two–way ranging (SS TWR) examples from the iLabs UWB library:

- **Node A (initiator)**: `ex_06a_ss_twr_initiator.ino`[8]
- **Node B (responder)**: `ex_06b_ss_twr_responder.ino`[9]

The responder replies to each poll frame, and the initiator computes the time-of-flight and prints the resulting distance in meters on the serial console (Figure 8).

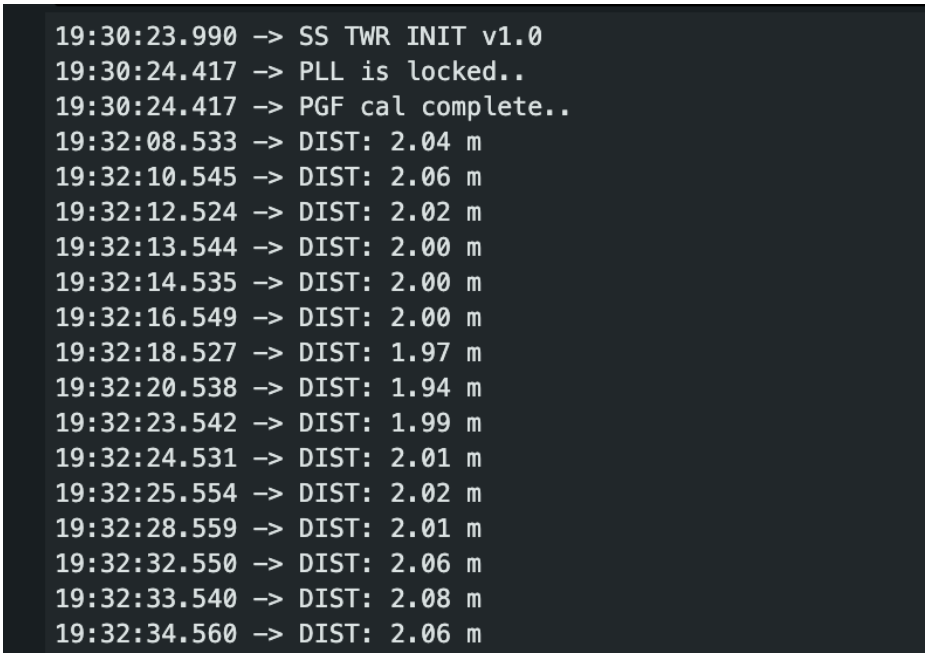

Figure 8: Serial output of `ex_06a_ss_twr_initiator.ino` showing repeated distance estimates between the two nodes.

[8] https://github.com/PontusO/iLabs_UWB/blob/main/examples/ex_06a_ss_twr_initiator/ex_06a_ss_twr_initiator.ino

[9] https://github.com/PontusO/iLabs_UWB/blob/main/examples/ex_06b_ss_twr_responder/ex_06b_ss_twr_responder.ino

6 Plan for Calibration and Data Logging

The goal of this section is to set up and document a complete workflow for collecting and analysing UWB distance measurements between one anchor and one tag. In the following subsections we will:

Objectives of this section

1. **Create a Python data-logging script** that records all received distance measurements together with precise timestamps and stores them in CSV files.
2. **Build a reproducible physical setup** where the two UWB nodes are mounted at the same height above the ground and placed at defined distances (e.g. 1 m, 2 m, 4 m, 8 m, 16 m).
3. **Run calibration experiments** for different configured sampling rates and distances, and save the raw data as `time_ms, distance_uwb_m` CSV files.
4. **Process the CSV data in Python** to compute, for each setup: the actual measurement frequency (Hz), mean distance and standard deviation, and compare these values with the real distances and configured sampling rates.

The next subsections describe these steps one by one: Python logger, physical measurement setup, experiment runs, and offline data analysis.