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

1 Programming Options for the Challenger RP2040 UWB **2**
1.1 RP2040 + DWM3000: Board Overview . . . 2
1.2 Firmware Options on the RP2040 . . . 2
1.3 Library Support for the DWM3000 . . . 2
1.4 Conclusion for This Project . . . 3

2 Arduino-Based Workflow for the Challenger RP2040 UWB **4**
2.1 Following the iLabs Getting-Started Guide . . . 4
2.2 Bootloader Issue: "No Drive Detected" . . . 4
2.3 Programming via UF2 Files . . . 5
2.3.1 One-time setup in the Arduino IDE . . . 5
2.3.2 Put the board into BOOTSEL mode . . . 5
2.3.3 Upload the program via UF2 . . . 5
2.4 FreeRTOS Configuration Error . . . 5
2.4.1 Enable FreeRTOS for the Challenger RP2040 UWB . . . 6

3 Which Data Can We Get from the DWM3000? **7**
3.1 Raw data directly from the DWM3000 . . . 7
3.2 Derived data on RP2040 using the UWB library . . . 7

4 Retrieving Raw UWB Data on the RP2040 **8**
4.1 How to run the raw-data demo . . . 8
4.2 Example serial output . . . 8
4.3 Sketch: `rx_raw_data_demo.ino` . . . 9

5 Example: Distance Measurement Between Two Nodes **10**

6 Plan for Calibration and Data Logging **11**
6.1 Python data logger and UWB initiator setup . . . 11
6.2 Physical UWB measurement setup . . . 12
6.3 Calibration runs for different target sampling rates . . . 13
6.4 Processing the CSV data in Python . . . 14
6.5 Accuracy, Precision and Effective update rate . . . 17

List of Figures

1 Conceptual view of the Challenger RP2040 UWB: RP2040 MCU, DWM3000 UWB module and their firmware options. . . . 2
2 Selecting the `iLabs Challenger 2040 UWB` board and USB serial port in the Arduino IDE. . . . 4
3 Typical Arduino upload error when the RP2040 UF2 drive is not detected. . . . 4
4 Compilation error when FreeRTOS is not enabled for the Challenger RP2040 UWB. . . . 5
5 * . . . 8
6 * . . . 8
7 Serial output of the TX example (left) and the raw-data receiver demo (right). . . . 8

8 Serial output of `ex_06a_ss_twr_initiator.ino` showing repeated distance estimates between the two nodes. . . . 10
9 Mechanical mounting of the Challenger RP2040 UWB boards for reproducible antenna–to–antenna distances. . . . 12
10 Complete measurement setup: initiator on the box, responder on the table, tape measure for distance settings, and laptop running the `uwb_logger.py` script. . . . 13
11 Plots for distance 1 m at all configured target sampling rates. . . . 15
12 Plots for distance 2 m at all configured target sampling rates. . . . 15
13 Plots for distance 4 m at all configured target sampling rates. . . . 16
14 Plots for distance 8 m at all configured target sampling rates. . . . 16
15 Plots for distance 12 m at all configured target sampling rates. . . . 17

List of Tables

1 Typical raw data available from the DWM3000 . . . 7
2 Examples of derived data on the RP2040 . . . 7
3 Generated calibration files for each distance and target sampling rate (file names without `.csv`) 14
4 Summary of target vs. measured update rate and distance statistics for all calibration runs. . . 18

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.

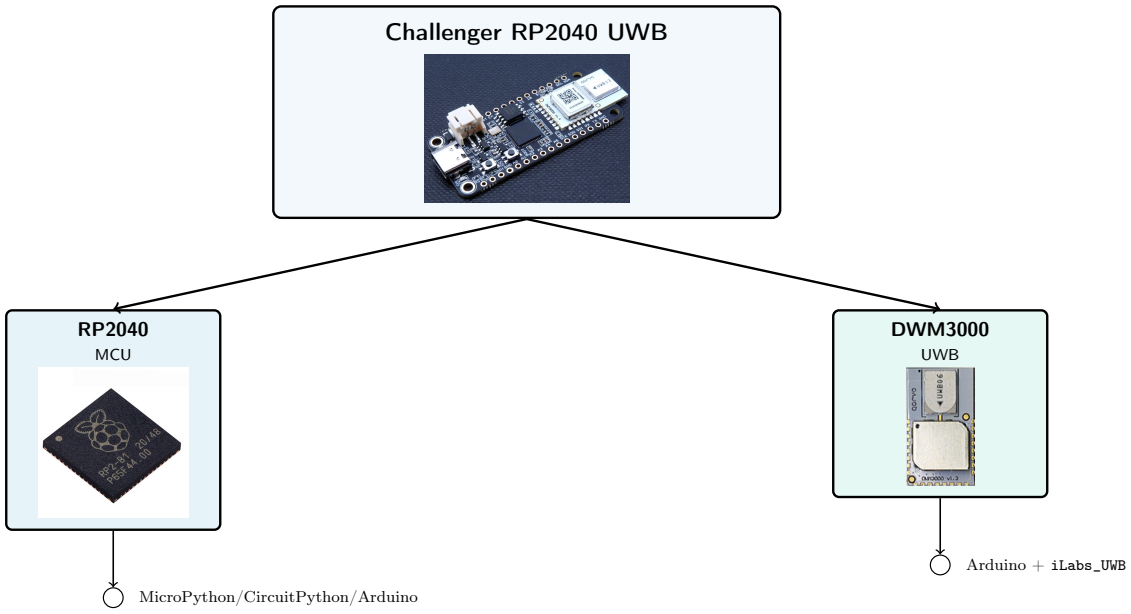


Figure 1: Conceptual view of the Challenger RP2040 UWB: RP2040 MCU, DWM3000 UWB module and their firmware options.

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.

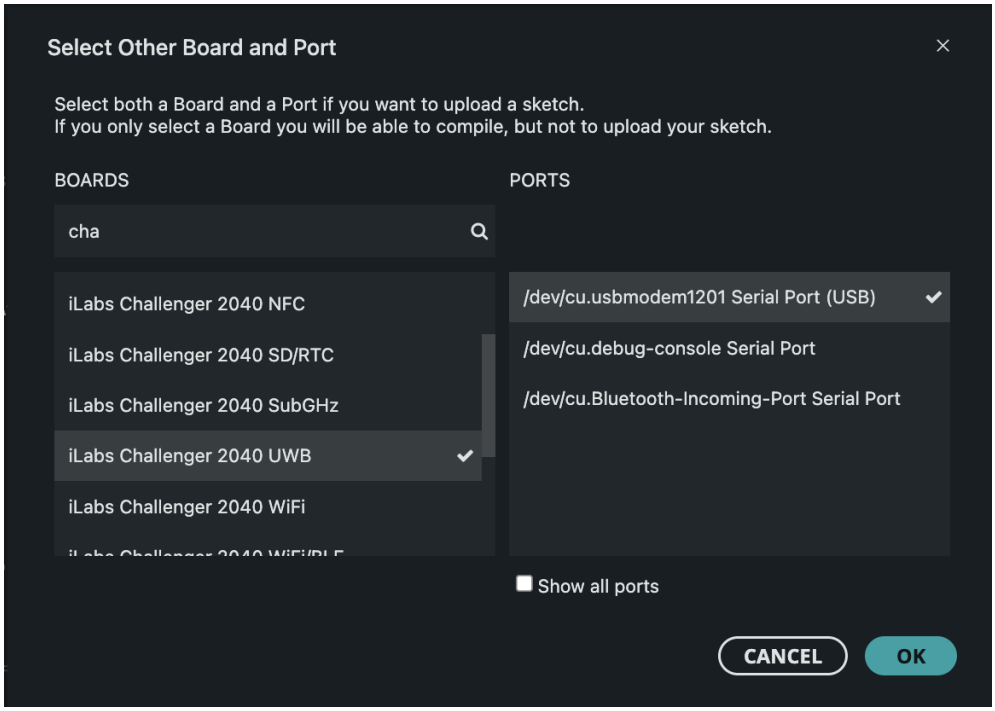


Figure 2: Selecting the `iLabs Challenger 2040 UWB` board and USB serial port in the Arduino IDE.

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:

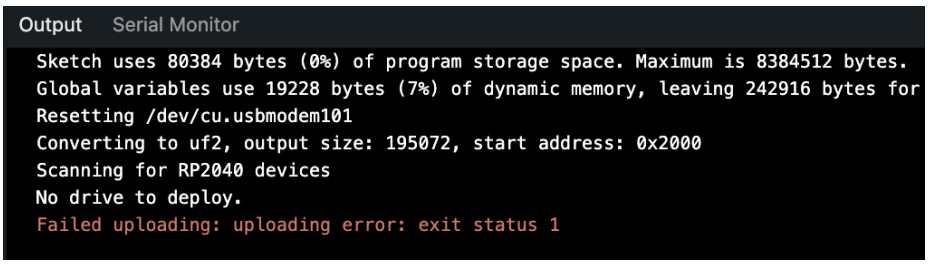


Figure 3: Typical Arduino upload error when the RP2040 UF2 drive is not detected.

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.

- **Fram grabber**: `rx_raw_data_demo.ino`[8]

[8] https://github.com/siddharthpatelde/Challenger-RP2040-UWB---Arduino-/blob/main/rx_raw_data_demo/rx_raw_data_demo.ino

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`[9]
- **Node B (responder)**: `ex_06b_ss_twr_responder.ino`[10]

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).

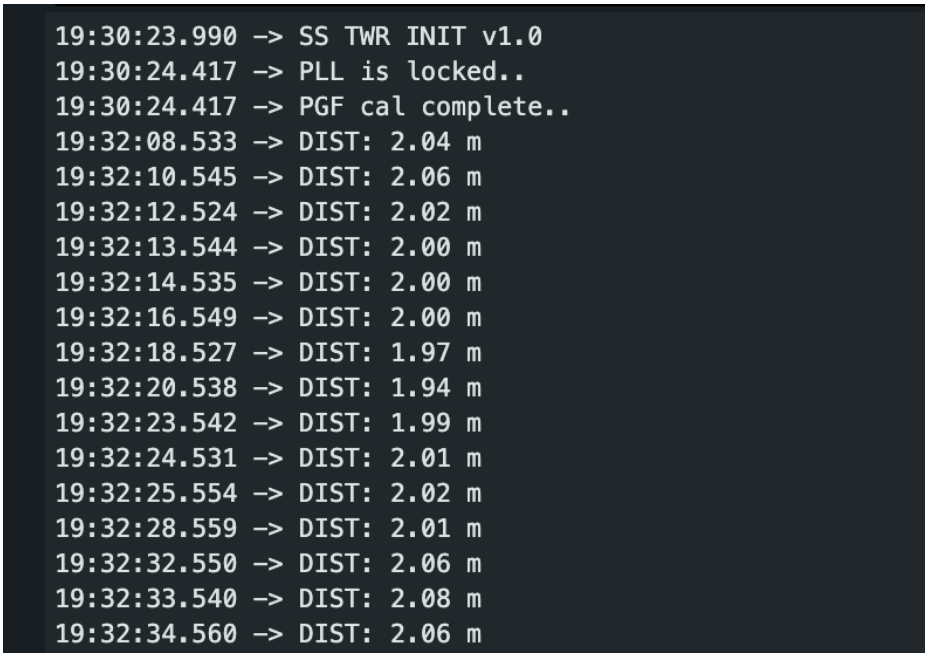


Figure 8: Serial output of `ex_06a_ss_twr_initiator.ino` showing repeated distance estimates between the two nodes.

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

[10] 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.

6.1 Python data logger and UWB initiator setup

In order to log the data as described, a simple data publishing `.ino` file was first created for the Challenger RP2040 UWB module, based on the single-sided TWR example code.

The new initiator sketch can be found here:[11]

UWB initiator and responder setup

1. Load `ss_twr_init.ino` on the first UWB Challenger module. This module acts as the **initiator** and outputs distance values.
2. On the second UWB module, load the example responder sketch:[a] `ex_06b_ss_twr_responder.ino`. This module acts as the **responder**.

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

After this setup is running, the distance values from the initiator are logged using a Python script.

The Python file is called `uwb_logger.py` and is located in the `/data_logger` folder of the GitHub repository (link not fixed here, because the exact path may change in the future).

[11] https://github.com/siddharthpatelde/Challenger-RP2040-UWB---Arduino-/blob/main/data_logger/ss_twr_init/ss_twr_init.ino

Python data logger configuration

In `uwb_logger.py`, set at least:

- the serial port where the UWB initiator is connected,
- the number of samples to collect.

Example configuration:

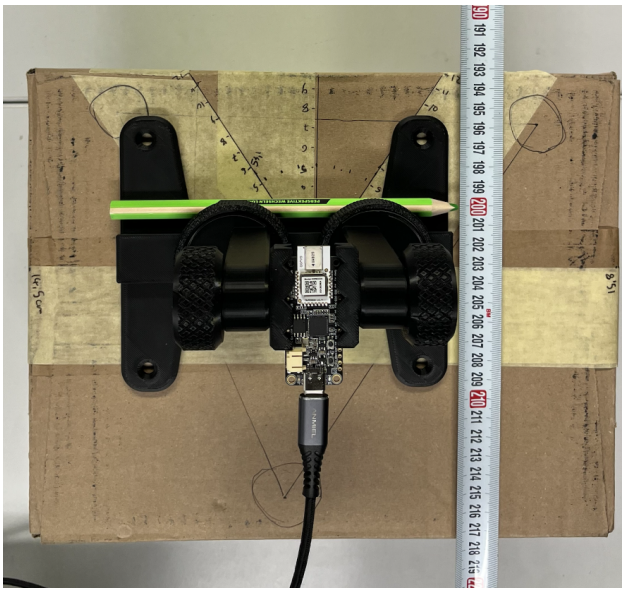
```
SERIAL_PORT        = '/dev/cu.usbmodemXXXX'
BAUD_RATE          = 115200
SAMPLES_TO_COLLECT = 1000
```

After running `uwb_logger.py` with these parameters, the logged data (distances with timestamps) is automatically saved into a CSV file with the chosen output filename.

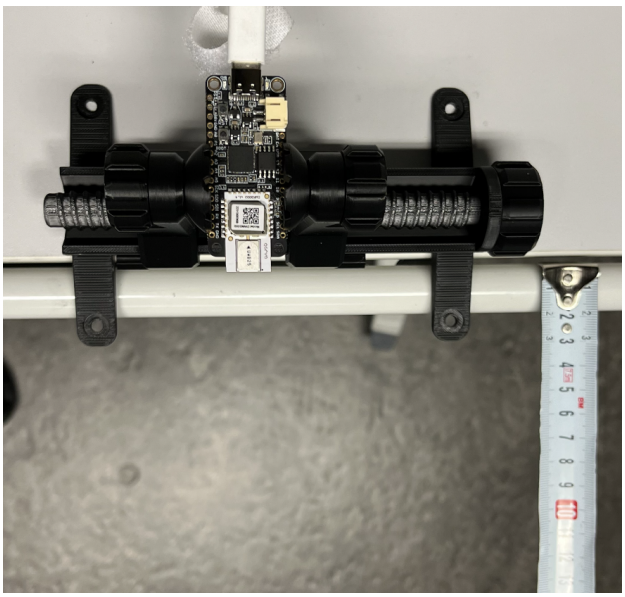
6.2 Physical UWB measurement setup

Now that the distance values can be logged over time, a stable mechanical setup was built so that both UWB modules are mounted at the same height and their *antenna–to–antenna distance* can be adjusted precisely.

Both boards are fixed in 3D-printed PCB holders. The holders are attached to the table (responder) and to a cardboard box (initiator) so that the antennas are aligned horizontally. With this setup the distance can be set reproducibly to $1\,\text{m}$, $2\,\text{m}$, $4\,\text{m}$, $8\,\text{m}$ and $12\,\text{m}$. (As will be shown later, at $16\,\text{m}$ separation no reliable data was received in this configuration.)



UWB initiator mounted in a 3D-printed holder on a cardboard box, antenna aligned with the tape measure.



UWB responder mounted at the same height on the table edge, again with a 3D-printed holder.

Figure 9: Mechanical mounting of the Challenger RP2040 UWB boards for reproducible antenna–to–antenna distances.

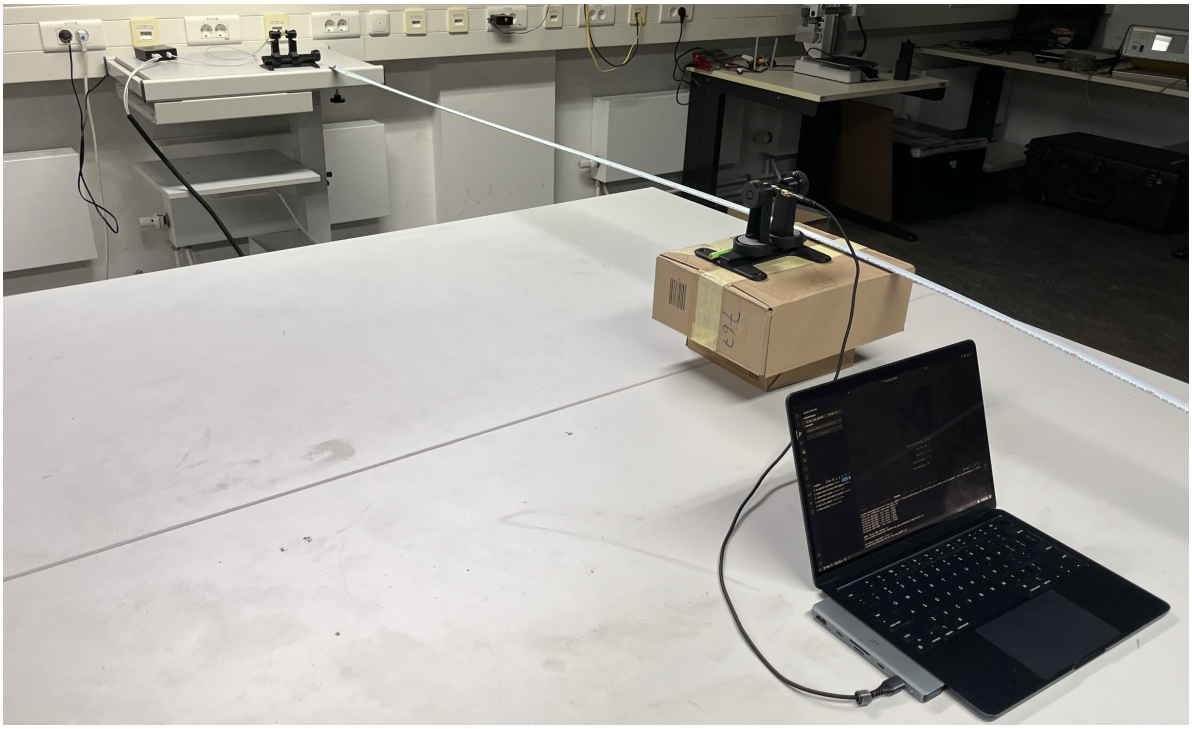


Figure 10: Complete measurement setup: initiator on the box, responder on the table, tape measure for distance settings, and laptop running the `uwb_logger.py` script.

6.3 Calibration runs for different target sampling rates

To run the calibration experiments we use

- the Python logger `uwb_logger.py` from Section 6.1, and
- the physical setup from Section 6.2.

The question is: *how do we change the frequency of the distance measurements?* This is done directly in the UWB initiator sketch `ss_twr_init.ino` by adjusting the inter-ranging delay:

```
/* Inter-ranging delay period, in milliseconds. */
#define RNG_DELAY_MS 16.7
// 10 Hz  = 100 ms
// 20 Hz  =  50 ms
// 40 Hz  =  25 ms
// 60 Hz  =  16.7 ms
// 80 Hz  =  12.5 ms
// 100 Hz =  10 ms
// 200 Hz =   5 ms
// 1 kHz  =   1 ms
// 10 kHz =   0.1 ms
```

For each combination of distance and target sampling rate, the logger is run until the configured number of samples is collected. The CSV files are stored in the `/data_logger/CSVs` folder of the GitHub repository.[12]

The naming scheme is:

[12] https://github.com/siddharthpatelde/Challenger-RP2040-UWB---Arduino-/tree/main/data_logger/CSVs

<XX>m_<XX>Hz.csv

Table 3 shows all generated filenames (file extension .csv omitted for brevity).

Table 3: Generated calibration files for each distance and target sampling rate (file names without .csv)

Distance / Rate	10 Hz	20 Hz	40 Hz	60 Hz	80 Hz	100 Hz	200 Hz	1 kHz	10 kHz
1 m	1m_10Hz	1m_20Hz	1m_40Hz	1m_60Hz	1m_80Hz	1m_100Hz	1m_200Hz	1m_1000Hz	1m_10000Hz
2 m	2m_10Hz	2m_20Hz	2m_40Hz	2m_60Hz	2m_80Hz	2m_100Hz	2m_200Hz	2m_1000Hz	2m_10000Hz
4 m	4m_10Hz	4m_20Hz	4m_40Hz	4m_60Hz	4m_80Hz	4m_100Hz	4m_200Hz	4m_1000Hz	4m_10000Hz
8 m	8m_10Hz	8m_20Hz	8m_40Hz	8m_60Hz	8m_80Hz	8m_100Hz	8m_200Hz	8m_1000Hz	8m_10000Hz
12 m	12m_10Hz	12m_20Hz	12m_40Hz	12m_60Hz	12m_80Hz	12m_100Hz	12m_200Hz	12m_1000Hz	12m_10000Hz

It is important to note that the sampling rate configured in the Arduino sketch (e.g. 100 Hz) does *not* guarantee that we will actually receive data at exactly this frequency. Packet loss, processing overhead on the microcontroller and other timing effects can reduce or jitter the effective data rate. Therefore we record data for different distances and target rates and later compare:

- the **target** frequency (from RNG_DELAY_MS),
- the **actual** frequency computed from timestamps in the CSV, and
- the **measured** distances versus the real physical distances.

6.4 Processing the CSV data in Python

Now that all CSV files contain time stamps and distances in a consistent format, they are processed with the Python script plot_uwb_logs.py in the data_logger folder.[13]

For each CSV file the script

- loads the data,
- computes mean distance, standard deviation and effective sampling rate,
- creates a plot (measured distance vs. time and summary statistics),
- and saves it as a PNG image in data_logger/plots/ with the same base name as the CSV file.

A typical call looks like this:

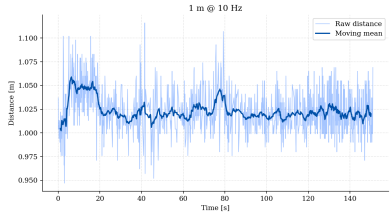
```
python plot_uwb_logs.py data_logger/CSVs
```

The resulting plots are named like the CSV files, e.g.

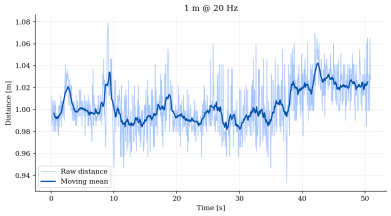
1m_10Hz.csv → 1m_10Hz.png

Below, all plots are grouped by distance. Each page shows the nine target sampling rates for one distance in a 3×3 grid; the caption under each plot is simply its file name.

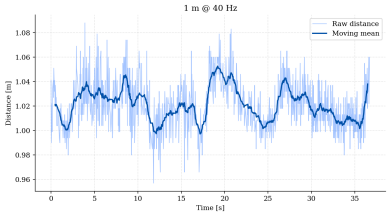
[13] https://github.com/siddharthpatelde/Challenger-RP2040-UWB---Arduino-/tree/main/data_logger



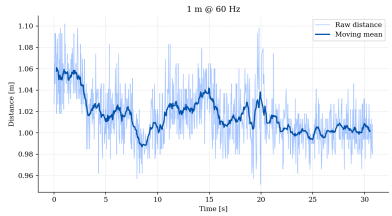
1m_10Hz.png



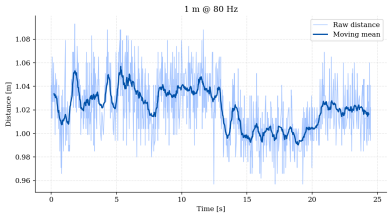
1m_20Hz.png



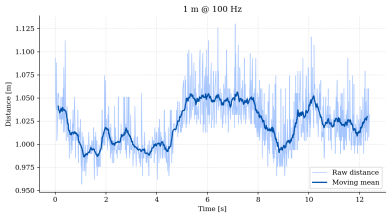
1m_40Hz.png



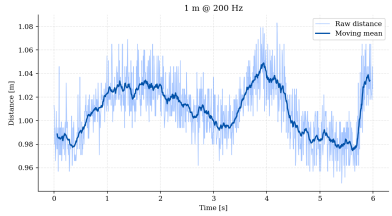
1m_60Hz.png



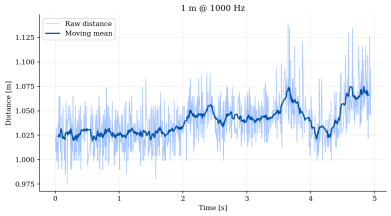
1m_80Hz.png



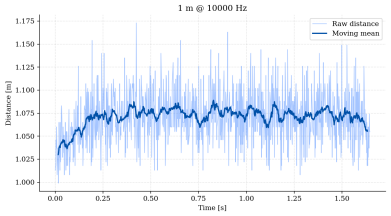
1m_100Hz.png



1m_200Hz.png

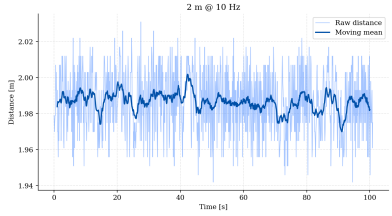


1m_1000Hz.png

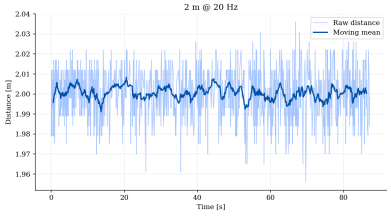


1m_10000Hz.png

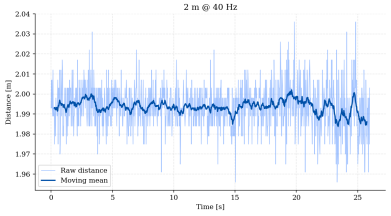
Figure 11: Plots for distance 1 m at all configured target sampling rates.



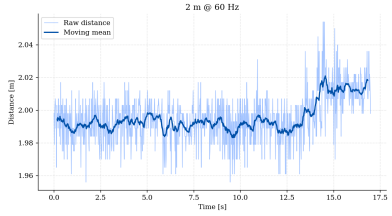
2m_10Hz.png



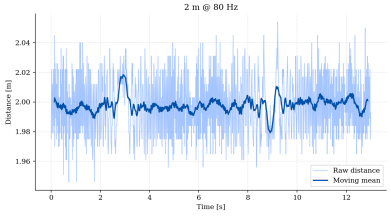
2m_20Hz.png



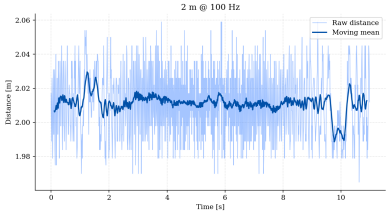
2m_40Hz.png



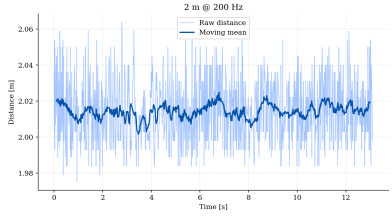
2m_60Hz.png



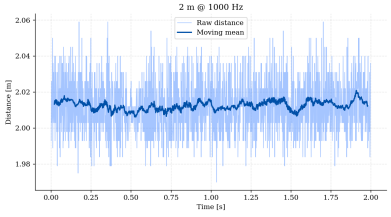
2m_80Hz.png



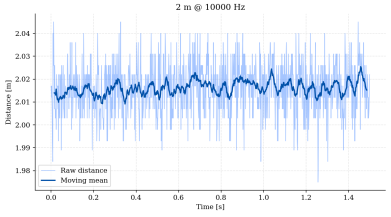
2m_100Hz.png



2m_200Hz.png

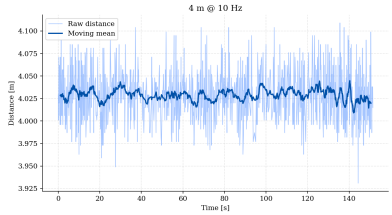


2m_1000Hz.png

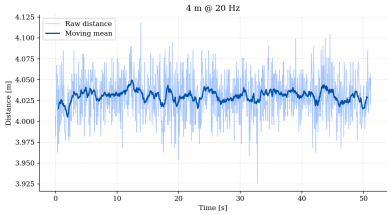


2m_10000Hz.png

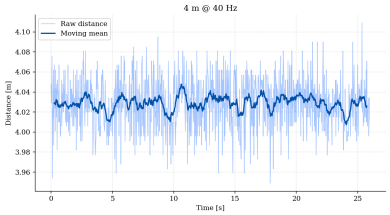
Figure 12: Plots for distance 2 m at all configured target sampling rates.



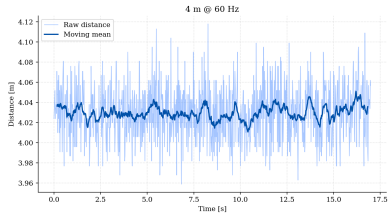
4m_10Hz.png



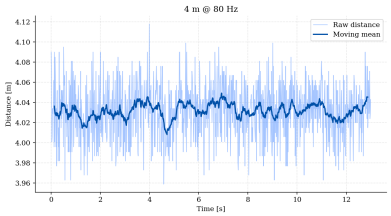
4m_20Hz.png



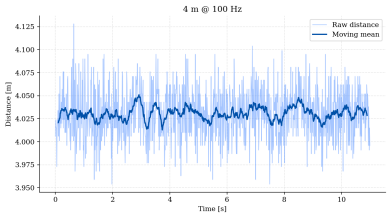
4m_40Hz.png



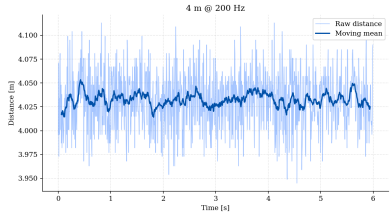
4m_60Hz.png



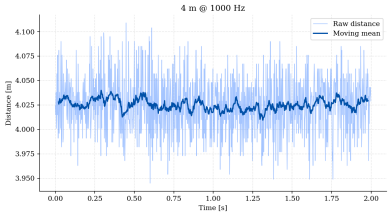
4m_80Hz.png



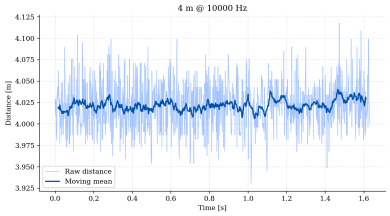
4m_100Hz.png



4m_200Hz.png

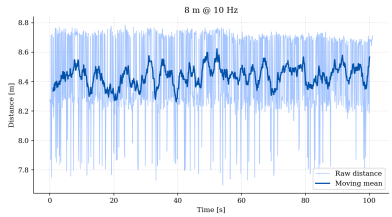


4m_1000Hz.png

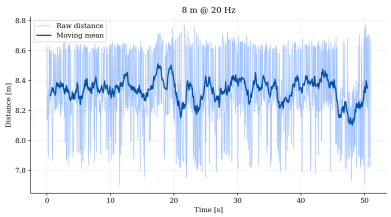


4m_10000Hz.png

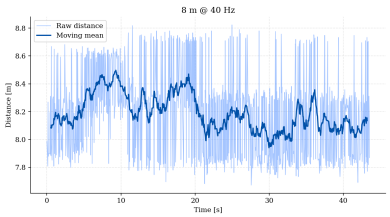
Figure 13: Plots for distance $4\,\mathrm{m}$ at all configured target sampling rates.



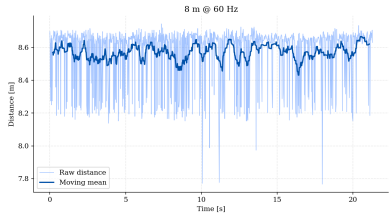
8m_10Hz.png



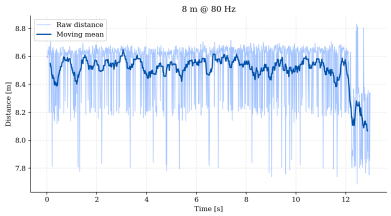
8m_20Hz.png



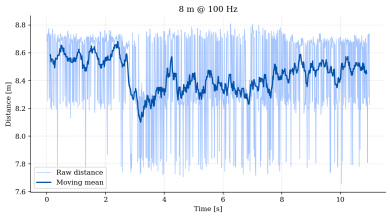
8m_40Hz.png



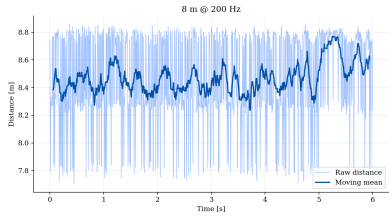
8m_60Hz.png



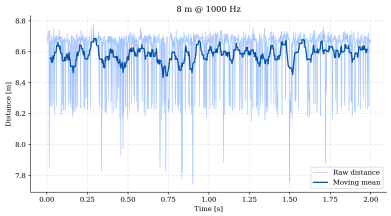
8m_80Hz.png



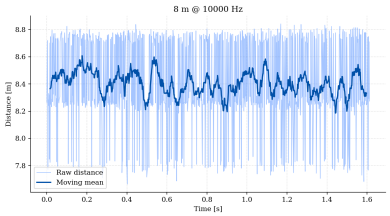
8m_100Hz.png



8m_200Hz.png

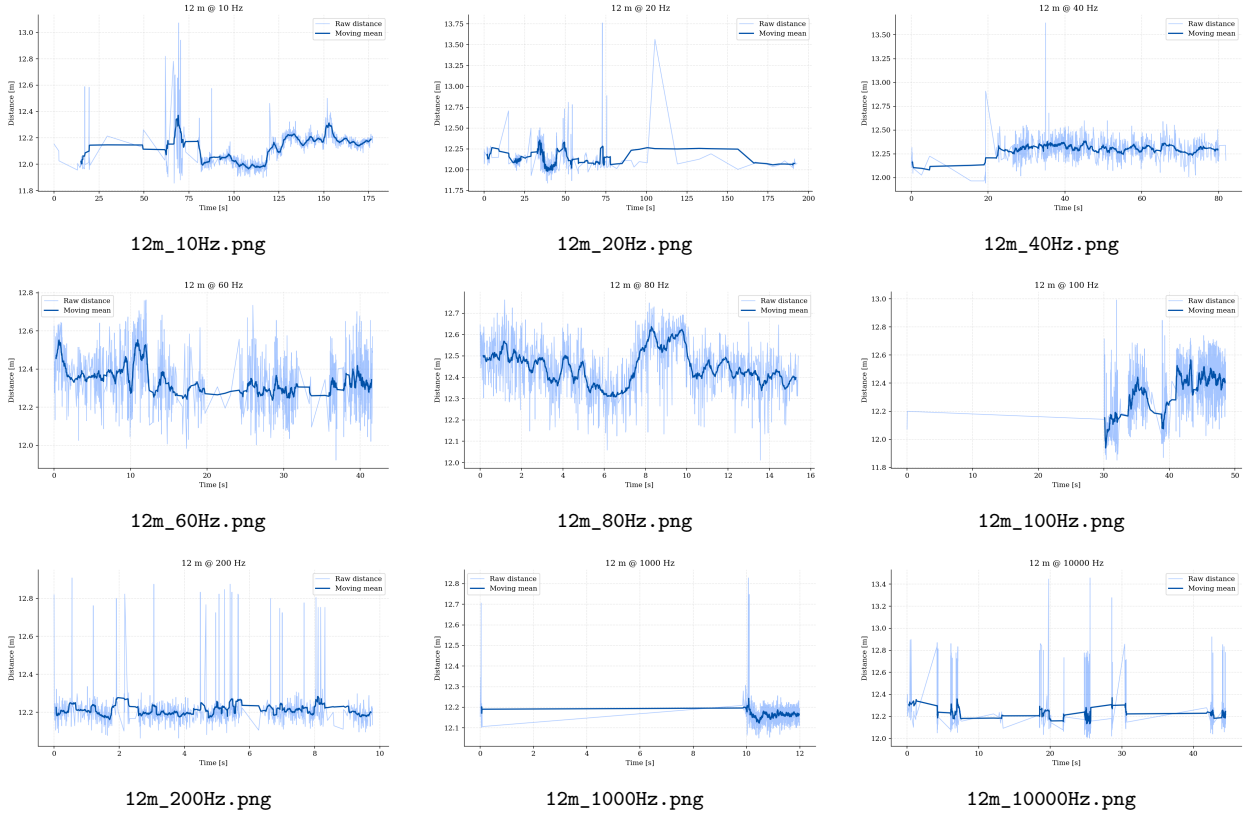


8m_1000Hz.png



8m_10000Hz.png

Figure 14: Plots for distance $8\,\mathrm{m}$ at all configured target sampling rates.



12m_10Hz.png 12m_20Hz.png 12m_40Hz.png

12m_60Hz.png 12m_80Hz.png 12m_100Hz.png

12m_200Hz.png 12m_1000Hz.png 12m_10000Hz.png

Figure 15: Plots for distance 12 m at all configured target sampling rates.

6.5 Accuracy, Precision and Effective update rate

Recalling the original goal (Tasks 1.c and 2.b), the recorded UWB logs were used to evaluate

- the **effective measurement frequency** f_{meas},
- the **accuracy** of the distance estimate (mean error), and
- the **precision** (spread of the measurements).

Each CSV file contains a time stamp t_i in milliseconds and a measured distance d_i in metres, with typically $N = 1000$ samples.

- **Actual update rate f_{meas}:** For consecutive samples the time difference is

$$\Delta t_i = t_i - t_{i-1} \,. \tag{1}$$

With time stamps in milliseconds, the effective update rate is computed as the inverse of the average time step:

$$f_{\text{meas}} = \frac{1000}{\frac{1}{N-1}\sum_{i=2}^{N} \Delta t_i} \quad [\text{Hz}] \,. \tag{2}$$

- **Accuracy (mean error):** The arithmetic mean of all measured distances is

$$\bar{d} = \frac{1}{N}\sum_{i=1}^{N} d_i \,. \tag{3}$$

The systematic error with respect to the ground-truth distance d_{target} is

$$\text{Error} = \bar{d} - d_{\text{target}} \,. \tag{4}$$

- **Precision (standard deviation):** The spread of the measurements is described by the sample standard deviation

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(d_i - \bar{d})^2} \,. \tag{5}$$

All metrics were computed automatically in Python using the script `analyze_uwb_data.py` in the `data_logger` folder.[14] The resulting summary table is stored as `UWB_Analysis_Summary.csv` in the same repository.[15]

Table 4 shows the evaluation results for all recorded runs, grouped by distance and target sampling rate.

Table 4: Summary of target vs. measured update rate and distance statistics for all calibration runs.

File	Target [Hz]	Actual [Hz]	Mean dist [m]	Std dev [m]
12m_10000Hz	10000.0	22.45	12.2191	0.1595
12m_1000Hz	1000.0	83.28	12.1634	0.0480
12m_100Hz	100.0	20.56	12.3539	0.1956
12m_10Hz	10.0	5.64	12.1230	0.1242
12m_200Hz	200.0	102.19	12.2089	0.0996
12m_20Hz	20.0	2.52	12.1197	0.1674
12m_40Hz	40.0	12.20	12.3065	0.1107
12m_60Hz	60.0	24.00	12.3444	0.1470
12m_80Hz	80.0	64.86	12.4500	0.1250
1m_10000Hz	10000.0	606.93	1.0705	0.0277
1m_1000Hz	1000.0	202.06	1.0393	0.0238
1m_100Hz	100.0	80.53	1.0215	0.0288
1m_10Hz	10.0	6.62	1.0235	0.0236
1m_200Hz	200.0	166.67	1.0088	0.0253
1m_20Hz	20.0	19.63	1.0039	0.0227
1m_40Hz	40.0	27.21	1.0209	0.0196
1m_60Hz	60.0	32.45	1.0171	0.0258
1m_80Hz	80.0	40.81	1.0220	0.0254
2m_10000Hz	10000.0	666.44	2.0160	0.0111
2m_1000Hz	1000.0	500.00	2.0125	0.0177
2m_100Hz	100.0	90.91	2.0112	0.0200
2m_10Hz	10.0	9.90	1.9866	0.0167
2m_200Hz	200.0	76.24	2.0145	0.0174
2m_20Hz	20.0	11.43	2.0004	0.0127
2m_40Hz	40.0	38.46	1.9936	0.0112
2m_60Hz	60.0	58.82	1.9954	0.0146
2m_80Hz	80.0	76.92	1.9979	0.0209
4m_10000Hz	10000.0	612.51	4.0219	0.0276
4m_1000Hz	1000.0	500.00	4.0253	0.0288
4m_100Hz	100.0	90.91	4.0305	0.0274
4m_10Hz	10.0	6.60	4.0287	0.0281
4m_200Hz	200.0	166.67	4.0324	0.0298
4m_20Hz	20.0	19.51	4.0304	0.0259
4m_40Hz	40.0	38.46	4.0287	0.0251
4m_60Hz	60.0	58.82	4.0298	0.0261
4m_80Hz	80.0	76.92	4.0318	0.0266
8m_10000Hz	10000.0	619.73	8.4034	0.3304
8m_1000Hz	1000.0	500.00	8.5824	0.1965
8m_100Hz	100.0	90.91	8.4391	0.3112
8m_10Hz	10.0	9.89	8.4376	0.2829
8m_200Hz	200.0	166.67	8.4667	0.3338
8m_20Hz	20.0	19.61	8.3350	0.2629
8m_40Hz	40.0	22.94	8.1871	0.3061
8m_60Hz	60.0	46.90	8.5691	0.1791
8m_80Hz	80.0	76.92	8.5040	0.2358

[14] https://github.com/siddharthpatelde/Challenger-RP2040-UWB---Arduino-/blob/main/data_logger/analyze_uwb_data.py

[15] https://github.com/siddharthpatelde/Challenger-RP2040-UWB---Arduino-/blob/main/data_logger/UWB_Analysis_Summary.csv