LABORATOIRE DE ROBOTIQUE

FOLLOW ME

DEVELOPMENT AND VALIDATION OF A UWB-BASED FOLLOW-ME ROBOTIC SYSTEM

NADER RIZK, JEREMY BONTE, THIBAULT VERNY

25th June 2026

PROJECT OVERVIEW

This Bachelor's thesis is conducted at the **Laboratoire d'Automatique (LA) at EPFL**, under the supervision of Dr. Christophe Salzmann. The project focuses on the development of a high-precision Indoor Positioning System (IPS) for autonomous mobile robotics.

The main objective is to design and implement a system capable of enabling a mobile platform to autonomously follow a user, similarly to the concept of a "smart suitcase" that tracks and follows its owner, hence the name of the project "Follow-me". This requires accurate, real-time estimation of the relative position between the user and the robot in indoor environments.

To achieve this, the system relies on Ultra-Wideband (UWB) technology, using Decawave DW3000 transceivers in combination with ESP32 microcontrollers. UWB is particularly well-suited for this application due to its ability to provide centimeter-level distance measurements based on precise time-of-flight (ToF) estimation of radio signals.

The developed system is based on a two-way ranging communication protocol between a mobile tag, carried by the user, and an anchor mounted on the robot. By continuously exchanging signals and measuring propagation times, the system estimates the distance between the two devices in real time. This information can then be used as an input for control algorithms that enable the robot to maintain a desired distance and follow the user smoothly.

Overall, this project combines embedded systems, wireless communication, and real-time signal processing to build a robust and accurate positioning solution, forming a key building block for autonomous "Follow-me" robotic applications.

ESP32-based UWB development board integrating the Decawave DW3000 transceiver for high-precision ranging.

PART 1

UWB 3000 SETUP

1.1 INTRODUCTION

This first part of the project presents the development and validation of the Ultra-Wideband (UWB) ranging system, aiming to establish reliable communication and accurate distance measurements between devices.

The work began with a minimal configuration of one tag and one anchor, which allowed the implementation of the poll-response two-way ranging process and the identification of key parameters: transmission delays, reception windows, and timeout durations. Early experiments revealed the system's sensitivity to timing misconfigurations, often resulting in missed packets and frequent timeouts.

The system was then extended to a configuration with one tag and two anchors, introducing additional challenges related to synchronization, increased communication load, and potential signal interference. This step provides a more realistic framework for future positioning applications, which require multiple anchors.

The main challenge was to achieve a compromise between measurement accuracy and communication robustness: raw measurements were affected by noise and occasional aberrant values (multipath effects, imperfect calibration, environmental disturbances), while overly strict timing parameters caused frequent timeouts and data loss.

A significant portion of the work therefore focused on tuning system parameters—reception delays, timeout windows, and hardware settings—to ensure both stable communication and consistent distance estimation. The outcome is a functional and robust ranging system that produces reliable measurements with a low communication failure rate, forming a solid basis for further development in positioning and control.

1.2 ULTRA-WIDEBAND RANGING PRINCIPLE

Ultra-Wideband (UWB) is a wireless communication technology that enables highly accurate distance measurement by transmitting very short-duration pulses over a wide frequency spectrum. Unlike traditional narrowband systems, UWB provides precise timing resolution, which is essential for accurate localization.

The system operates using a two-way ranging (TWR) protocol, based on a poll-response exchange between a tag and an anchor. The tag first sends a poll message, which is received by the anchor. After a known processing delay, the anchor transmits a response message back to the tag.

By measuring the timestamps of transmission and reception, the system estimates the signal's time-of-flight (ToF). In the case of single-sided two-way ranging (SS-TWR), the ToF can be approximated as:

$$\mathrm{ToF} = \frac{T_{\text{round}} - T_{\text{reply}}}{2} \tag{1.1}$$

where:

- T_{round} is the total round-trip time measured by the tag,
- T_{reply} is the processing delay at the anchor.

The distance between the tag and the anchor is then computed using:

$$d = c \cdot \mathrm{ToF} \tag{1.2}$$

where c is the speed of light.

This method allows for distance estimation with centimeter-level accuracy. However, its performance depends strongly on precise timing calibration, including antenna delays and properly configured transmission and reception windows. Environmental factors such as noise and multipath propagation can also introduce errors, making filtering and parameter tuning essential for reliable measurements.

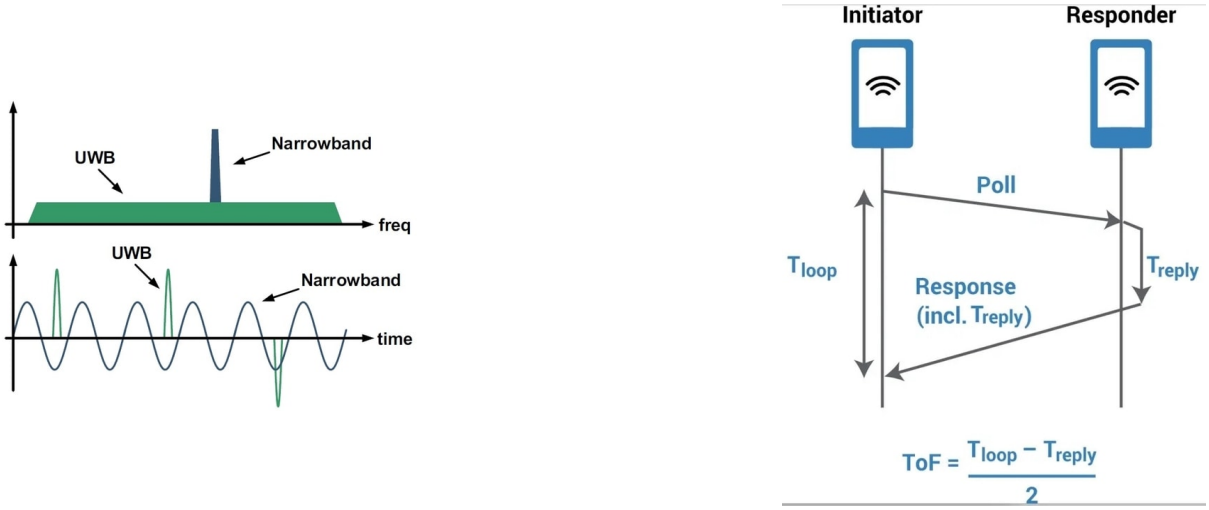


FIGURE 1.1
Comparison between Ultra-Wideband (UWB) and narrowband signals in the frequency and time domains (left), and illustration of the two-way ranging (TWR) process used to estimate the time-of-flight (ToF) and compute distance (right).

1.3 INITIAL SETUP: SINGLE TAG AND SINGLE ANCHOR

The initial phase of the project was based on a minimal configuration consisting of one tag and one anchor, using ESP32 microcontrollers coupled with DW3000 Ultra-Wideband (UWB) transceivers. This setup was chosen to validate the basic communication process and to establish a reliable distance measurement framework.

In this configuration, the tag initiates the communication by transmitting a poll message to the anchor. Upon reception, the anchor processes the message and sends back a response. The tag then uses the timestamps of transmission and reception to compute the time-of-flight (ToF) of the signal, which is directly related to the distance between the two devices.

This simple architecture allowed for the identification and tuning of key system parameters, including transmission delays, reception windows, and timeout durations. It also highlighted several practical challenges, such as communication timeouts, unstable measurements, and sensitivity to timing misalignment.

Overall, this first setup served as a fundamental validation step, providing a controlled environment to understand the behavior of the UWB system before extending it to more complex multi-anchor configurations.

1.4 FINAL SETUP: TWO ANCHORS AND ONE TAG

This new configuration represents a significant step toward a full positioning system, as multiple anchors enable spatial awareness and future position estimation through triangulation.

In this setup, the tag sequentially communicates with each anchor using a poll-response two-way ranging protocol. Each anchor independently receives the poll message, processes it, and transmits a response after a predefined delay. The tag then computes the distance to each anchor based on the measured time-of-flight (ToF).

1.5 ARDUINO CODE SETUP

This section presents the Arduino implementation of the UWB ranging system used in the two-anchor, one-tag configuration. The code is split into two main parts: the tag code, which initiates the ranging exchange and computes the distance, and the anchor code, which receives the tag message and sends back a timed response. Both programs rely on the DW3000 library and must share the same radio configuration parameters in order to communicate correctly.

At a high level, the ranging process follows a single-sided two-way ranging protocol. The tag sends a poll message to one anchor, waits for the corresponding response, and computes the distance from the timing information embedded in the exchanged frames. The process is then repeated for the second anchor. This sequential polling strategy avoids collisions between anchors and makes it possible to recover two independent distance measurements using a single mobile tag.

A major part of the code is dedicated to the configuration of the DW3000 transceiver. Parameters such as the channel number, preamble length, data rate, antenna delays, reception timeout, and transmission delays are essential because they directly affect whether frames are correctly sent, received, and interpreted. The final performance of the system therefore depends not only on the ranging formula itself, but also on the proper tuning of these low-level communication parameters.

1.5.1 TAG CODE

The tag code is responsible for initiating the ranging process, alternating between the two anchors, collecting the responses, and computing the measured distances.

The first important part of the program is the hardware setup:

```
#define PIN_RST 27
#define PIN_IRQ 34
#define PIN_SS  4
```

These definitions specify the physical connection between the ESP32 and the DW3000 module. The reset pin allows the module to be restarted, the interrupt pin is used by the DW3000 for status signaling, and the slave-select pin is required for SPI communication. These pins must match the actual wiring of the board, otherwise the module cannot be initialized correctly.

Another key block of the code is the set of timing constants:

```
#define RNG_DELAY_MS               50
#define TX_ANT_DLY                 16385
#define RX_ANT_DLY                 16385
#define POLL_TX_TO_RESP_RX_DLY_UUS 500
#define RESP_RX_TIMEOUT_UUS        1500
```

These constants govern the overall timing of the ranging cycle. The parameter `RNG_DELAY_MS` sets the delay between two successive measurements and therefore controls the measurement frequency. In this version, a value of 50 ms corresponds to an approximate update rate of about 20 Hz, before accounting for the additional time spent in communication and processing.

The parameters `TX_ANT_DLY` and `RX_ANT_DLY` compensate for the internal delays introduced by the antenna and RF circuitry. These values are crucial for accurate ranging because the DW3000 measures timestamps at the chip level, while the signal physically enters and exits the system through the antenna. Without these corrections, all measured distances would contain a systematic bias.

The parameter `POLL_TX_TO_RESP_RX_DLY_UUS` defines how long the tag waits after transmitting a poll before enabling reception. This delay must be large enough to allow the anchor to receive the poll, process it, and prepare its response. If it is too short, the tag may start listening before the anchor is ready; if it is too long, the beginning of the anchor response may be missed. The parameter `RESP_RX_TIMEOUT_UUS` defines how long the tag waits for a response before declaring a timeout. This value was one of the critical tuning parameters in the project because it directly affects the trade-off between robustness and responsiveness.

The DW3000 radio configuration is set through:

```
static dwt_config_t config = {
    5, DWT_PLEN_128, DWT_PAC8, 9, 9, 1,
    DWT_BR_6M8, DWT_PHRMODE_STD, DWT_PHRRATE_STD,
    (129 + 8 - 8), DWT_STS_MODE_OFF, DWT_STS_LEN_64, DWT_PDOA_M0
};
```

This structure contains the fundamental radio parameters shared by both the tag and the anchors. The channel number is set to 5, meaning that all devices communicate on the same UWB channel. This parameter must be identical on all boards. The preamble length, preamble acquisition chunk size, preamble codes, and SFD mode influence synchronization and detection reliability. The data rate is set to `DWT_BR_6M8`, corresponding to a high-rate configuration that allows fast packet exchanges. Since all

devices must interpret the same physical-layer frame format, these values must match exactly between tag and anchors.

The setup function initializes the DW3000 and applies all of these parameters:

```
dwt_configure();
dwt_configuretxrf();
dwt_setrxantennadelay();
dwt_settxantennadelay();
dwt_setrxaftertxdelay();
dwt_setrxtimeout();
dwt_setlnapamode();
```

Each of these functions plays an important role. The function `dwt_configure()` applies the main communication parameters. The function `dwt_configuretxrf()` sets the transmission RF parameters. The functions `dwt_setrxantennadelay()` and `dwt_settxantennadelay()` apply the delay compensation discussed previously. The function `dwt_setrxaftertxdelay()` defines when the tag opens its receive window after sending a poll, and `dwt_setrxtimeout()` defines how long that receive window remains open. Finally, `dwt_setlnapamode()` enables the low-noise amplifier and power amplifier, which improves reception sensitivity and transmission quality.

The tag then alternates between the two anchors:

```
static const uint8_t ANCHORS[][2] = {{'A','1'},{'A','2'}};
static uint8_t anchor_idx = 0;
```

This is an important design choice. Rather than attempting to communicate with both anchors at the same time, the tag polls them one after the other. This sequential strategy reduces the risk of packet collisions and simplifies synchronization. At each loop iteration, the current anchor address is inserted into the outgoing frame, and the tag expects a response only from that anchor.

The message addressing is handled with:

```
tx_poll_msg[5] = anchor[0]; tx_poll_msg[6] = anchor[1];
tx_poll_msg[7] = TAG_ADDR[0]; tx_poll_msg[8] = TAG_ADDR[1];
```

This part is essential because it ensures that each poll message is directed to the correct anchor. The expected response frame is also updated accordingly, so that the tag can verify that the received message indeed comes from the anchor it has just polled.

The actual poll transmission is performed with:

```
dwt_writetxdata();
dwt_writetxfctrl();
dwt_starttx();
```

These functions load the poll message into the DW3000 transmit buffer, configure the frame length, and start immediate transmission. The flag `DWT_RESPONSE_EXPECTED` is particularly important because it informs the DW3000 that a response is anticipated, allowing the receiver to be enabled automatically according to the configured delay.

The program then waits for one of three outcomes:

```
SYS_STATUS_RXFCG_BIT_MASK
SYS_STATUS_ALL_RX_TO
SYS_STATUS_ALL_RX_ERR
```

A good frame reception, a timeout, or a reception error. This distinction is important experimentally because it allows the user to determine whether poor results come from missing responses or from corrupted receptions.

If a valid response is received, the tag extracts four timestamps:

- the poll transmission timestamp at the tag,
- the response reception timestamp at the tag,
- the poll reception timestamp at the anchor,
- the response transmission timestamp at the anchor.

These are read using:

```
poll_tx_ts = dwt_readtxtimestamplo32();
resp_rx_ts = dwt_readrxtimestamplo32();
resp_msg_get_ts(&rx_buffer[RESP_MSG_POLL_RX_TS_IDX], &poll_rx_ts);
resp_msg_get_ts(&rx_buffer[RESP_MSG_RESP_TX_TS_IDX], &resp_tx_ts);
```

This is the core of the ranging system. The anchor embeds its own timestamps in the response message, and the tag combines them with its local timestamps to estimate the time of flight.

The distance is then computed using:

```
int32_t rtd_init = (int32_t)(resp_rx_ts - poll_tx_ts);
int32_t rtd_resp = (int32_t)(resp_tx_ts - poll_rx_ts);
double tof = ((rtd_init - rtd_resp * (1.0f - clockOffsetRatio)) / 2.0)
             * DWT_TIME_UNITS;
double distance_m = tof * SPEED_OF_LIGHT;
```

This formula corresponds to single-sided two-way ranging. The term `rtd_init` is the round-trip duration measured at the tag, while `rtd_resp` is the reply duration measured at the anchor. The clock offset ratio compensates for the fact that the clocks of the tag and anchor are not perfectly identical. After applying the correction, the time of flight is converted into a distance using the speed of light.

The final part of the loop stores the last measured distance for each anchor and prints the result after the **second** anchor has been polled:

```
if (anchor[1] == '2') {
    Serial.print("DIST;");
    Serial.print(lastA1, 2);
    Serial.print(";");
    Serial.println(lastA2, 2);
}
```

This output format is convenient because each printed line contains the latest available distances to both anchors. If one measurement fails because of a timeout or reception error, the corresponding value is replaced with `NaN`, making communication failures easy to detect in MATLAB or during later analysis.

1.5.2 ANCHOR CODE

The anchor code complements the tag code by waiting for a poll frame, recording its reception timestamp, and transmitting a delayed response containing the timing information required for distance computation.

As in the tag code, the first important elements are the hardware definitions:

```
#define PIN_RST 27
#define PIN_IRQ 34
#define PIN_SS  4
```

These values must again match the actual ESP32–DW3000 wiring. The anchor also uses antenna delay compensation:

```
#define TX_ANT_DLY 16387
#define RX_ANT_DLY 16387
```

The fact that these values differ slightly from those used on the tag is important. Antenna delays are hardware-dependent and may not be identical across boards. In practice, each board may require its own calibration. This is one of the key parameters to calibrate if ever distances seem too far off their real distance.

One of the most important timing constants in the anchor is:

```
#define POLL_RX_TO_RESP_TX_DLY_UUS 850
```

This parameter defines the delay between receiving the poll and sending the response. It is a central parameter in the entire system. It must be large enough to allow the anchor to process the received frame, prepare the timestamps, and schedule the delayed transmission correctly. At the same time, it should not be unnecessarily large, because that would increase the overall measurement cycle duration. This value was therefore determined experimentally as a compromise between reliable response generation and efficient operation.

The anchor uses the same DW3000 radio configuration as the tag:

```
static dwt_config_t config = {
5, DWT_PLEN_128, DWT_PAC8, 9, 9, 1, DWT_BR_6M8, DWT_PHRMODE_STD,
DWT_PHRRATE_STD, (129 + 8 - 8), DWT_STS_MODE_OFF, DWT_STS_LEN_64, DWT_PDOA_M0
};
```

This matching is mandatory. If the channel, preamble settings, or data rate differ from those of the tag, the anchor will not decode the incoming poll correctly.

The anchor frame definitions are also important:

```
static uint8_t rx_poll_msg[] = { ... ANC0, ANC1, TAG0, TAG1, 0xE0, ... };
static uint8_t tx_resp_msg[] = { ... TAG0, TAG1, ANC0, ANC1, 0xE1, ... };
```

These arrays define the expected structure of the incoming poll frame and the structure of the outgoing response. The addressing fields allow the anchor to verify that the poll is intended for it, which is especially important in a multi-anchor setup.

During setup, the anchor initializes the DW3000, applies the configuration, and enables the LNA/PA mode:

```
dwt_configure();
dwt_configuretxrf();
dwt_setrxantennadelay();
dwt_settxantennadelay();
dwt_setlnapamode();
```

Unlike the tag, the anchor does not configure a response-expected receive mode, because its role is not to initiate exchanges but to stay ready to receive a poll.

The anchor main loop begins with:

```
dwt_rxenable(DWT_START_RX_IMMEDIATE);
```

This line is essential because it explicitly places the anchor in receive mode. From that point onward, the anchor waits until a good frame or a reception error is detected.

When a valid poll is received, the anchor first stores the poll reception timestamp:

```
poll_rx_ts = get_rx_timestamp_u64();
```

This timestamp is one of the four time values needed for the single-sided two-way ranging formula. The anchor then computes the future transmission time of the response:

```
resp_tx_time = (poll_rx_ts + (POLL_RX_TO_RESP_TX_DLY_UUS * UUS_TO_DWT_TIME)) >>
dwt_setdelayedtrxtime(resp_tx_time);
```

This is one of the most important parts of the anchor code. Rather than sending the response immediately, the anchor schedules a delayed transmission at a precisely defined instant. This deterministic timing is necessary because the tag needs to know, either directly or indirectly, how much time elapsed at the anchor between poll reception and response transmission.

The response transmission timestamp is then reconstructed with:

```
resp_tx_ts = (((uint64_t)(resp_tx_time & 0xFFFFFFFEUL)) << 8) + TX_ANT_DLY;
```

This calculation includes the antenna delay correction. The resulting timestamp represents the actual physical transmission instant used in the ranging formula.

The anchor then inserts both timestamps into the response frame:

```
resp_msg_set_ts(&tx_resp_msg[RESP_MSG_POLL_RX_TS_IDX], poll_rx_ts);
resp_msg_set_ts(&tx_resp_msg[RESP_MSG_RESP_TX_TS_IDX], resp_tx_ts);
```

This is the core information that the tag later uses to compute the distance. Without these embedded timestamps, the initiator would only know its own round-trip time and would not be able to separate propagation delay from anchor processing delay.

Finally, the anchor sends the response using:

```
dwt_writetxdata(sizeof(tx_resp_msg), tx_resp_msg, 0);
dwt_writetxfctrl(sizeof(tx_resp_msg), 0, 1);
ret = dwt_starttx(DWT_START_TX_DELAYED);
```

The use of `DWT_START_TX_DELAYED` is particularly important. It ensures that the response is transmitted at the scheduled instant rather than immediately, which makes the timing repeatable and compatible with the tag's ranging computation.

If transmission succeeds, the code waits for the transmission-finished flag and then increments the sequence number. If a reception error occurs, the anchor simply clears the corresponding status bits and returns to listening mode.

Overall, the anchor code is simpler than the tag code because it does not compute distances itself. Its role is to act as a precisely timed responder. Nevertheless, its timing parameters and delayed-transmission mechanism are fundamental, since any error in the anchor response schedule directly degrades the quality of the ranging measurements or leads to timeouts at the tag.

1.6 SPECIFIC SETUP

This section presents the specific parameter choices adopted in the final implementation and the experimental reasoning used to select them.

1.6.1 SELECTION OF `RNG_DELAY_MS`

The parameter `RNG_DELAY_MS` sets the delay introduced at the end of each ranging cycle in the tag code. In practice, it directly controls the interval between two successive measurements and therefore strongly influences the effective update rate of the system. A smaller value leads to more frequent ranging exchanges and better temporal resolution, while a larger value reduces the measurement rate and generally improves signal smoothness.

To determine an appropriate value for this parameter, several experimental tests were conducted using different delay values. For each test, the measured distances to the two anchors and the corresponding estimated radial speeds were plotted as functions of time. The aim was to identify the best compromise between responsiveness, measurement stability, and noise amplification.

For `RNG_DELAY_MS = 10`, the system operates at a very high update rate. The measured distances follow the motion of the tag with good reactivity, but the estimated radial speed exhibits very large fluctuations and unrealistic spikes. These spikes do not correspond to actual physical motion and indicate strong amplification of small measurement errors. This configuration is therefore too sensitive to noise and does not provide sufficiently stable derived measurements.

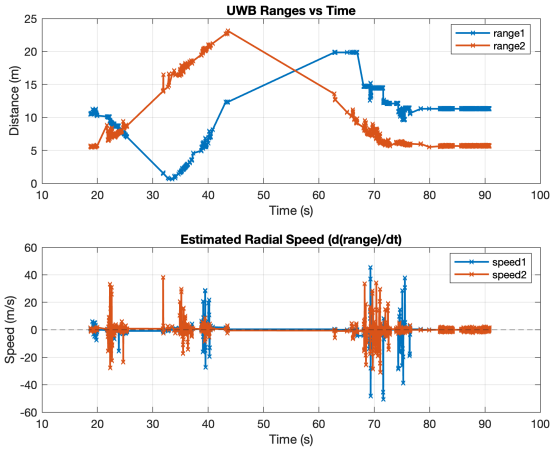


FIGURE 1.2
Distance and estimated radial speed obtained with `RNG_DELAY_MS = 10`.

For `RNG_DELAY_MS = 25`, the results are improved compared to the previous case. The distance curves remain responsive and the radial speed is less erratic, but important fluctuations are still present,

especially during transitions and in the more dynamic parts of the motion. Although this setting is more usable, the noise remains significant and would make control applications less robust.

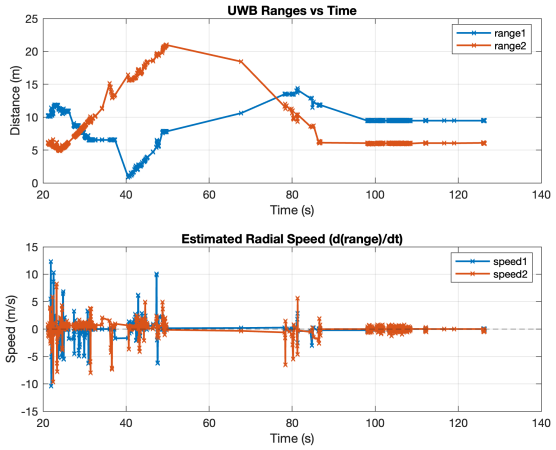


FIGURE 1.3
Distance and estimated radial speed obtained with `RNG_DELAY_MS = 25`.

For `RNG_DELAY_MS = 50`, the system reaches a much better compromise. The measured distances remain sufficiently smooth while still capturing the main variations of the tag motion. At the same time, the estimated radial speed becomes significantly more stable, with fluctuations of much smaller amplitude than in the previous cases. The main motion trends are still visible, but the amplification of noise is reduced to a level that is much more compatible with interpretation and later control use.

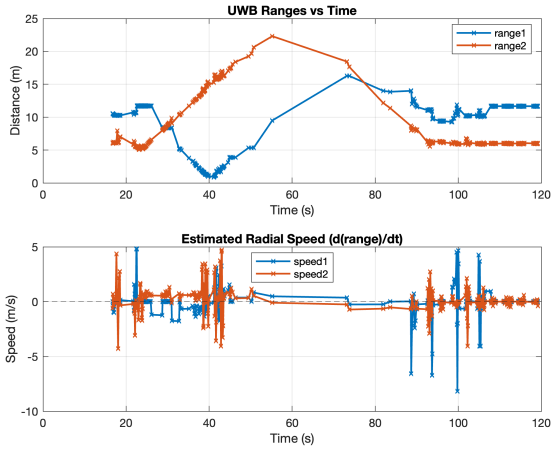


FIGURE 1.4
Distance and estimated radial speed obtained with `RNG_DELAY_MS = 50`.

For `RNG_DELAY_MS = 100`, the radial speed becomes even smoother and the noise level is reduced further. However, this comes at the cost of responsiveness. The distance curves become less reactive, and rapid changes in motion are represented more coarsely. In other words, the system becomes more stable but less suitable for dynamic tracking.

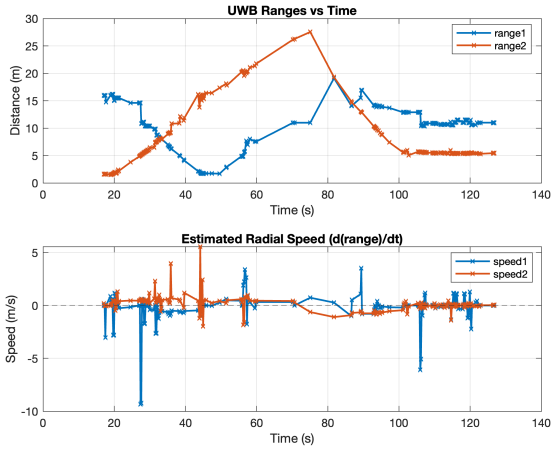


FIGURE 1.5
Distance and estimated radial speed obtained with `RNG_DELAY_MS = 100`.

These experiments clearly show the trade-off introduced by `RNG_DELAY_MS`. A small delay improves reactivity but amplifies noise, especially when computing quantities derived from distance such as radial speed. A large delay smooths the measurements but reduces temporal resolution and responsiveness.

Based on the observed results, `RNG_DELAY_MS = 50` was selected as the best compromise. This value provides a good balance between update rate and stability: the distance measurements remain representative of the motion, while the derived speed signal is considerably less noisy than for lower delay values. It therefore offers the most suitable operating point for the current stage of the project.

1.6.2 TEMPORARY RESULTS

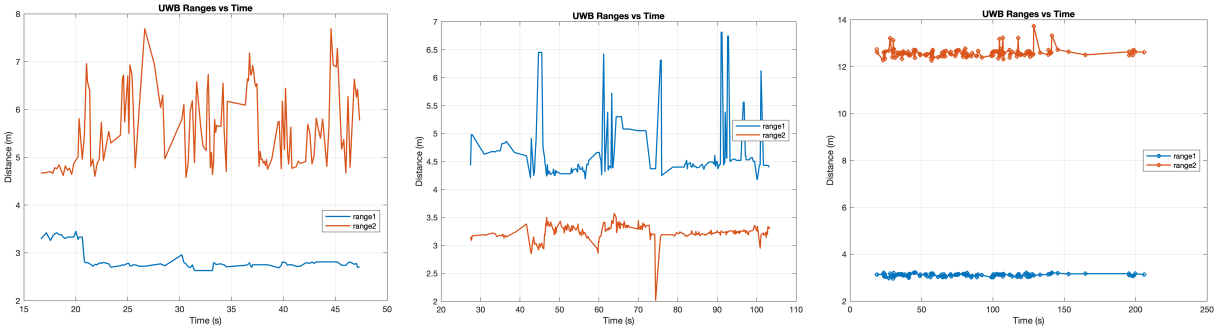
After selecting `RNG_DELAY_MS = 50`, the behavior of the system improved noticeably. The measurements became significantly less noisy than in the previous high-frequency tests, which confirmed that this value was a better compromise between responsiveness and stability. At this stage, additional tests were carried out by changing the physical arrangement of the modules, in particular by swapping the anchor modules and modifying their relative orientation. The objective was to verify whether the observed behavior was linked to the hardware itself, to the geometry of the setup, or to the communication timing.

Taken as a whole, the graphs show that the quality of the measurements depends not only on the code parameters, but also very strongly on the spatial configuration of the system. In some configurations, both measured distances remain remarkably stable over time. This is the case in the graphs where one anchor stays close to approximately 3 m while the other remains near 12.5–13 m, or where the two measured distances remain around 12.8 m and 3.4 m with only very small fluctuations. These results indicate that, under favorable orientation and placement conditions, the UWB system is capable of producing consistent and repeatable measurements.

However, other graphs clearly show that the measurement quality is not uniform across all configurations. In some cases, one anchor remains relatively stable while the other exhibits much larger fluctuations, with abrupt peaks or dips superimposed on the nominal value. For example, in one configuration, one measured distance stays near 2.7–3.0 m while the other oscillates strongly between approximately 4.8 m and more than 7 m. In another case, one anchor remains near 3.2 m while the other shows repeated spikes

up to more than 6 m. These behaviors suggest that the corresponding communication link was more sensitive to environmental effects such as multipath propagation or NLOS conditions, partial obstruction, or unfavorable antenna orientation.

The comparison between the different plots also shows that swapping the modules can change the measurement quality. This suggests that the anchors are not perfectly interchangeable in practice, even if they run similar code. Small differences in calibration, antenna delays, mounting conditions, or local surroundings may produce measurable changes in performance. The orientation of the modules also appears to be important: when the relative angle between the tag and anchor is less favorable, the signal quality degrades and the measured distance becomes more erratic. This is consistent with the known sensitivity of UWB ranging to line-of-sight conditions and antenna alignment.



(A) Configuration 1 (B) Configuration 2 (C) Configuration 3

(D) Configuration 4 (E) Configuration 5 (F) Dynamic mouvement

FIGURE 1.6

Comparison of UWB distance measurements for six different module placements and orientations after setting `RNG_DELAY_MS = 50`. The results illustrate the influence of the physical setup on measurement stability, noise, and the occurrence of communication irregularities.

Another important observation concerns the timeouts. In these plots, timeouts do not necessarily appear as explicit markers, but they can be inferred indirectly from discontinuities, missing transitions, flattened segments, or sudden jumps between two successive valid values. Since the code keeps the last valid distance for each anchor and only updates it when a correct response is received, a timeout can appear as a short plateau followed by an abrupt change once communication resumes. In more degraded configurations, the presence of isolated spikes and irregular updates also suggests that some ranging cycles were not completed successfully. In practice, this means that although `RNG_DELAY_MS = 50` reduced the effect of noise, it did not fully eliminate communication failures.

1.6.3 TIMEOUTS

One of the main difficulties encountered during this first phase of the project was the presence of frequent timeouts during the ranging exchanges. In practice, this meant that the tag transmitted a poll message but did not receive a valid response within the expected reception window. As a consequence, some ranging cycles were lost, measurements became discontinuous, and the recorded distances were either missing or corrupted. Finding a compromise between stable distance values and a sufficiently low timeout rate therefore became one of the most important challenges of this first part of the project.

The final solution was obtained by tuning the following DW3000 timing and calibration parameters:

```
dwt_setrxantennadelay(RX_ANT_DLY);
dwt_settxantennadelay(TX_ANT_DLY);
dwt_setrxaftertxdelay(POLL_TX_TO_RESP_RX_DLY_UUS);
dwt_setrxtimeout(RESP_RX_TIMEOUT_UUS);
```

These parameters are essential because they determine when the signal is considered to leave or enter the hardware, when the tag begins listening after transmitting a poll, and how long it keeps its reception window open. In other words, the system does not simply depend on sending and receiving packets: it depends on a precise temporal alignment between the poll transmission, the anchor processing time, the scheduled anchor response, and the opening of the reception window on the tag.

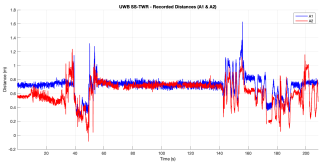
A timeout can therefore occur for several technical reasons. First, if the anchor response is scheduled too early or too late with respect to the tag reception window, the tag may simply miss the frame. Second, if the timeout window is too short, a valid response may arrive after the receiver has already stopped listening. Third, if timing corrections such as antenna delays are poorly chosen, the internal timestamps become less consistent and the overall exchange becomes less robust. Finally, weak reception conditions, multipath propagation, or unfavorable angles may further degrade packet detection and make the system more sensitive to timing misalignment.

To understand this behavior and identify the best compromise, several tests were carried out with different combinations of response delays and tag timeout values. The corresponding results are shown in Figures 1.7.

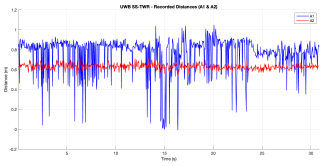
TEST 1: ANCH2 = 500, ANCH1 = 1000, POLL_TX_TO_RESP = 300, RESP_RX_TIMEOUT = 3000. This first configuration used two different anchor reply delays, with anchor 2 replying significantly earlier than anchor 1. The recorded distances show clear instability, with one anchor exhibiting large fluctuations and abrupt jumps. This indicates that the response timing was not well balanced across the two anchors. In particular, the anchor replying too early likely reduced the timing margin and made the reception more sensitive to synchronization errors and detection issues.

TEST 2: ANCH2 = 850, ANCH1 = 1000, POLL_TX_TO_RESP = 300, RESP_RX_TIMEOUT = 3000. Increasing the delay of anchor 2 improved the overall behavior compared to Test 1. However, the asymmetry between the two anchors still produced an uneven ranging performance. The graphs suggest that the system became more stable, but not yet fully consistent, as one link still appeared more sensitive to large spikes and temporary degradation.

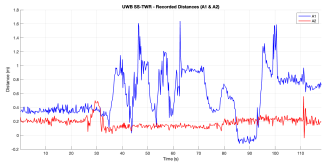
TEST 3: ANCH2 = 850, ANCH1 = 850, POLL_TX_TO_RESP = 300, RESP_RX_TIMEOUT = 3000. Using the same response delay on both anchors produced a clear improvement. The measurements became much more balanced and the two links behaved more consistently. This result shows that equalizing the anchor timing was important: when both anchors respond according to the same timing logic, the sequential ranging process becomes easier to synchronize and less prone to irregular behavior.



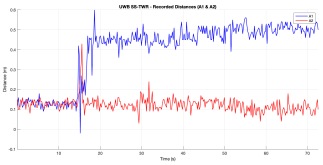
(A) Test 1: `A2=500, A1=1000, RX after TX=300, timeout=3000.`



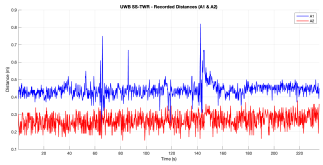
(B) Test 2: `A2=850, A1=1000, RX after TX=300, timeout=3000.`



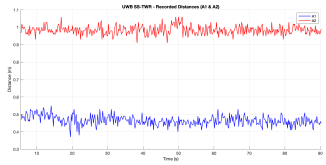
(C) Test 3: `A1=A2=850, RX after TX=300, timeout=3000.`



(D) Test 4: `A1=A2=850, RX after TX=300, timeout=5000.`



(E) Test 5: `A1=A2=850, RX after TX=300, timeout=1000.`



(F) Test 8: `A1=A2=850, RX after TX=500, timeout=1500.`

FIGURE 1.7
Comparison of the main timeout-related tests. The figures illustrate the influence of anchor response delay, tag receive-after-transmit delay, and receive timeout on distance stability and communication reliability.

TEST 4: ANCH2 = 850, ANCH1 = 850, POLL_TX_TO_RESP = 300, RESP_RX_TIMEOUT = 5000. Increasing the tag timeout further made the system more tolerant to delayed responses. In principle, this reduces the number of missed packets. However, the graphs show that this also allows more noisy or irregular receptions to be accepted. The measurements remain available, but the signal becomes less clean, with more fluctuations and less consistency. This confirms that a timeout window that is too long may improve robustness in terms of packet reception, but can degrade the quality of the measured data.

TEST 5: ANCH2 = 850, ANCH1 = 850, POLL_TX_TO_RESP = 300, RESP_RX_TIMEOUT = 1000. Reducing the timeout to 1000 produced the opposite behavior. The system became stricter, and the graphs indicate more abrupt interruptions, drops, and irregular updates. This suggests that valid responses were sometimes arriving after the receive window had already closed. The result is a cleaner acceptance criterion, but too many missed exchanges. The system therefore became less reliable in practice.

TEST 6: ANCH2 = 850, ANCH1 = 850, POLL_TX_TO_RESP = 300, RESP_RX_TIMEOUT = 50. Although no graph was retained for this case, this value is clearly too small for the system. A timeout of only 50 is insufficient for the complete ranging exchange, including anchor processing and radio propagation. This configuration therefore leads almost systematically to missed responses.

TEST 7: ANCH2 = 850, ANCH1 = 850, POLL_TX_TO_RESP = 300, RESP_RX_TIMEOUT = 2000. This case represents an intermediate compromise compared to Tests 4 and 5. It is more permissive than a 1000 timeout and less permissive than 3000 or 5000. Although more usable than the shortest timeout values, it still did not provide the best overall balance between low timeout rate and signal quality.

TEST 8: ANCH1 = 850, ANCH2 = 850, POLL_TX_TO_RESP = 500, RESP_RX_TIMEOUT = 1500. This final configuration gave the best overall results. The anchor response delays were kept

equal, which preserved symmetry and consistency between the two links. More importantly, the tag reception start delay was increased from 300 to 500. This gave the anchors more time to process the poll and prepare their delayed response before the tag opened its listening window. At the same time, the timeout was reduced to 1500, which kept the reception window sufficiently wide to capture valid responses while avoiding an unnecessarily long listening period. The graphs show that this configuration leads to the most stable and physically consistent measurements, with a lower apparent timeout rate and significantly fewer aberrant values.

These tests provide an important insight into the way the DW3000 modules operate. The communication problem is not only a question of whether the signal exists physically, but also of whether the receiver is listening at the correct moment and for the correct duration. In other words, the modules require precise temporal coordination. If the response is sent outside the expected interval, the tag interprets the situation as a timeout even though the anchor may have transmitted correctly. Conversely, if the listening window is too wide, the system becomes more tolerant but may also admit degraded or noisy measurements. The final values therefore reflect a compromise between strict timing and practical robustness.

Overall, these experiments showed that solving the timeout issue required understanding the DW3000 not just as a radio transceiver, but as a highly timing-sensitive ranging device. This was one of the main lessons of the first part of the project. By tuning the antenna delays, the receive-after-transmit delay, and the timeout window, it was possible to significantly reduce communication failures while preserving good measurement quality. The final selected values provided the most reliable operation and formed the basis for the rest of the work.

1.6.4 MATERIAL ATTENUATION TESTS UNDER OBSTRUCTED LINE-OF-SIGHT CONDITIONS

After the calibration of the UWB ranging system, a set of additional experiments was performed to evaluate the robustness of the distance measurements in the presence of obstacles. The objective of these tests was not to reproduce a fully non-line-of-sight scenario, where the direct path between the transmitter and receiver is completely blocked, but rather to study obstructed line-of-sight conditions. In these configurations, the direct propagation path may still exist, but it is partially attenuated, reflected, or disturbed by an object placed between the tag and the anchors.

This type of test is relevant for the Follow-me application, since the UWB signal may be affected by common obstacles during real operation. For example, the robot body, furniture, a laptop, or even the user can partially obstruct the propagation path between the tag and the anchors. Such obstructions can introduce a bias in the measured distance, increase the measurement noise, generate outliers, or produce temporary dropouts. Understanding these effects is therefore important before using the raw UWB distances as inputs for a control algorithm.

All tests were performed with the tag and the two anchors placed at a fixed reference distance of approximately $1.85\ \mathrm{m}$. The same calibrated UWB parameters were used for all acquisitions in order to ensure that the observed differences were mainly due to the obstacle configuration and not to a change in the ranging setup:

```
#define RNG_DELAY_MS                50
#define TX_ANT_DLY                  16350
#define RX_ANT_DLY                  16350
#define POLL_TX_TO_RESP_RX_DLY_UUS  500
#define RESP_RX_TIMEOUT_UUS         1500
```

The measurements were acquired through the serial interface and processed in MATLAB. For each configuration, the distances to anchor A1 and anchor A2 were recorded over time and saved as CSV files.

The resulting plots were then compared with a reference line-of-sight measurement obtained without any obstacle. The analysis focuses mainly on the stability of the measured distances, the presence of systematic bias, the occurrence of peaks or outliers, and the possible appearance of missing measurements.

The experimental study was divided into four configurations. First, a reference line-of-sight test was performed without any obstacle. Then, a plastic board was introduced either in front of the anchors or in front of the tag in order to evaluate the effect of a weakly attenuating material. A dynamic obstruction test was also carried out with a person passing between the tag and the anchors. Finally, a laptop was used as a stronger obstruction, representing an object containing metallic and electronic components. The corresponding photographs of the experimental setups are included in each subsection to document the position of the obstacle relative to the UWB modules.

REFERENCE LINE-OF-SIGHT TEST

The first acquisition was performed without any obstacle between the tag and the two anchors. This configuration was used as the reference line-of-sight case for the attenuation tests. The objective of this baseline measurement was to verify the stability of the calibrated UWB system under nominal propagation conditions before introducing any obstructing material.

During this test, the tag and the anchors were placed at a fixed distance of approximately 1.85 m. No object was intentionally placed on the propagation path, so the direct UWB signal between the devices was expected to be the dominant component. This setup is shown in Figure 1.8.

FIGURE 1.8
Reference line-of-sight configuration without obstacle between the tag and the anchors. The distance between the devices was approximately 1.85 m.

The recorded distances are shown in Figure 1.9. Both anchors remain close to the expected distance during

the acquisition. The measurements fluctuate slightly around the reference value, which is expected due to residual measurement noise, multipath reflections in the laboratory environment, and small differences in the orientation of the antennas. Anchor A1 exhibits slightly larger variations than A2. An isolated peak can be observed around 15 s, where the measured distance briefly drops below the nominal range. Since this deviation is not repeated and is not representative of the overall behavior, it is interpreted as an isolated outlier, most likely caused by a missed or corrupted ranging exchange. Apart from this isolated outlier, no sustained instability is observed.

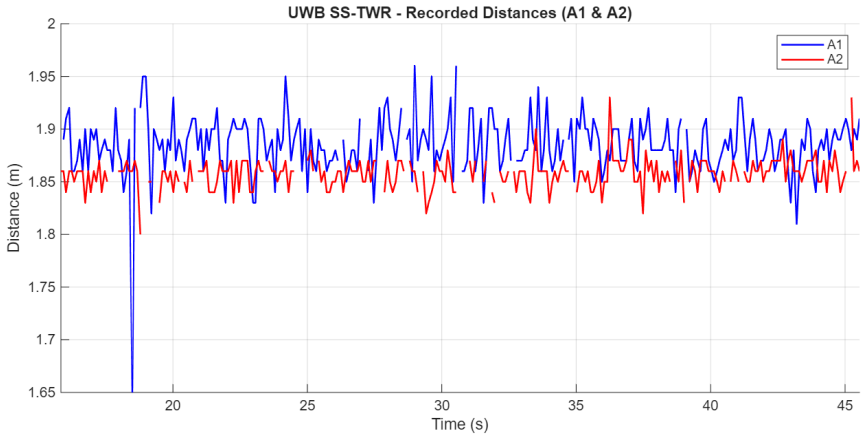


FIGURE 1.9
UWB distance measurements for the reference line-of-sight test. The measured distances remain close to the real distance of approximately 1.85 m, confirming the stability of the calibrated ranging setup without obstruction.

This reference acquisition confirms that the calibrated system provides consistent distance measurements when the propagation path is unobstructed. It also provides a baseline for comparing the following material attenuation tests. Any increase in bias, noise, outliers, or missing measurements observed in the obstructed configurations can therefore be attributed mainly to the presence of the tested obstacle.

PLASTIC BOARD ATTENUATION TEST

The first obstructed line-of-sight experiment was performed using a plastic board placed between the UWB devices. This test was designed to evaluate the effect of a weakly attenuating material on the ranging stability. Compared with metallic objects or the human body, plastic is expected to disturb the UWB signal only moderately. However, it can still introduce attenuation, reflections, or small changes in the propagation path.

Two configurations were tested. In the first configuration, the plastic board was placed in front of the anchors. In the second configuration, the same plastic board was placed in front of the tag. This made it possible to compare the influence of the obstacle position on the measured distances. The experimental setups are shown in Figure 1.10.

(a) Plastic board in front of the anchors

(b) Plastic board in front of the tag

FIGURE 1.10
Experimental configurations for the plastic board attenuation test. The board was successively placed in front of the anchors and in front of the tag while keeping the reference distance at approximately 1.85 m.

Figure 1.11 shows the distance measurements when the plastic board was placed in front of the anchors. Compared with the reference line-of-sight test, the measurements remain globally close to the expected distance of approximately 1.85 m. Anchor A2 stays relatively stable during most of the acquisition, while Anchor A1 shows larger fluctuations and several local peaks above the nominal distance. However, these deviations remain limited and do not indicate a sustained loss of communication or a strong systematic bias.

This result suggests that placing the plastic board near the anchors only weakly affects the UWB propagation. The main visible effect is a moderate increase in short-term variability, especially on A1, rather than a complete degradation of the ranging link. The signal remains usable for both anchors.

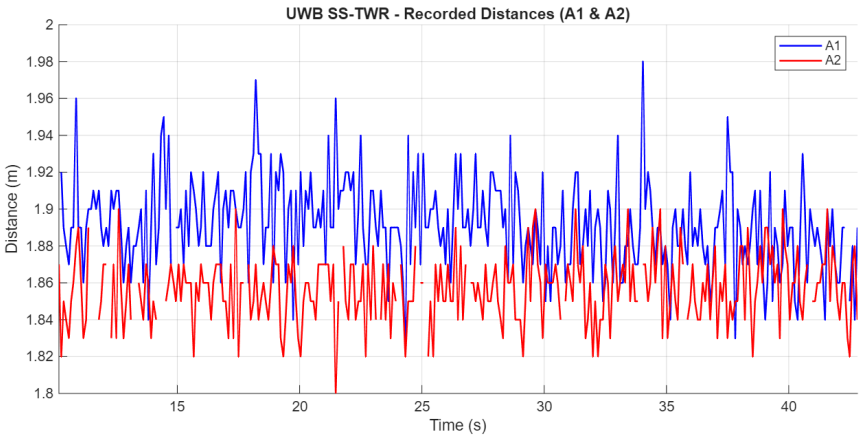


FIGURE 1.11
UWB distance measurements with the plastic board placed in front of the anchors. The measurements remain close to the reference distance, showing only moderate perturbations.

The second configuration, where the plastic board was placed in front of the tag, is shown in Figure 1.12. In this case, the measurements are still centered around the expected distance, but the fluctuations become more pronounced than in the anchor-side configuration. Anchor A1 exhibits a wider spread and several higher peaks, while Anchor A2 remains more stable but still shows occasional deviations.

The comparison between the two configurations indicates that the position of the plastic board has an influence on the ranging behavior. When the board is placed near the anchors, the disturbance remains moderate and mostly affects A1. When it is placed near the tag, the perturbation becomes more visible, probably because the obstacle is close to the device that initiates the ranging exchanges with both anchors. This can disturb both the transmitted poll messages and the received responses.

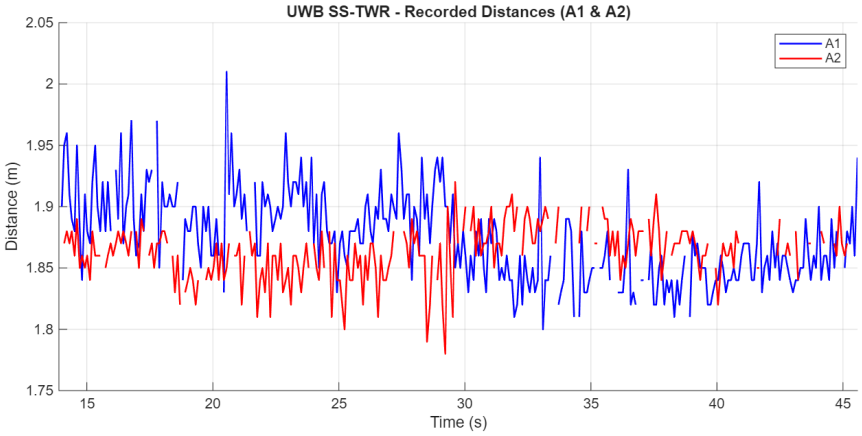


FIGURE 1.12
UWB distance measurements with the plastic board placed in front of the tag. The signal remains usable, but the measurements show slightly larger fluctuations compared with the reference case.

Overall, the plastic board does not strongly attenuate the UWB signal, since both anchors continue to provide measurements close to the reference distance. Nevertheless, the test shows that even a non-metallic obstacle can slightly increase measurement noise, especially when it is located close to the tag. This confirms that mild obstructed line-of-sight conditions remain manageable for the current setup, provided that the distance measurements are filtered before being used for control.

HUMAN-BODY DYNAMIC OBSTRUCTION TEST

A dynamic obstruction test was performed with a person standing and moving between the tag and the anchors at distinct time intervals during the acquisition. This test is particularly relevant for the Follow-me application, since the user or another person may temporarily obstruct the propagation path between the UWB modules during operation.

Compared with the previous static tests, this configuration introduces a time-varying obstruction. The human body can attenuate and reflect the UWB signal, mainly because it contains a high proportion of water and can partially block the direct propagation path. As a result, temporary disturbances, peaks, or missing measurements may appear when the person is located between the tag and the anchors.

The experimental setup is shown in Figure 1.13. During the acquisition, the person was intentionally placed between the tag and the anchors for short time intervals, then moved away from the propagation path. This makes it possible to observe how the ranging system reacts to the appearance and disappearance of a dynamic obstacle.

FIGURE 1.13
Human-body dynamic obstruction setup. A person was placed between the tag and the anchors at distinct time intervals in order to evaluate the effect of a temporary obstruction on the UWB distance measurements.

The recorded distances are shown in Figure 1.14. During the unobstructed intervals, the measured distances remain close to the reference value of approximately 1.85 m. The main disturbances appear

at approximately 26 s, 37 s, and 45 s, which correspond to the moments when the person was placed between the tag and the anchors. At these instants, the curves exhibit strong transient deviations from the nominal distance, with peaks reaching significantly higher values than in the unobstructed intervals. These variations are much larger than the normal measurement noise observed in the reference case.

These peaks indicate that the human body temporarily modifies the propagation conditions between the UWB modules. The direct path is partially obstructed, while reflected paths may become more significant. As a result, the measured time of flight can be distorted, leading to short overestimations or underestimations of the distance. However, the effect remains temporary: once the person moves away from the propagation path, the measurements rapidly return close to the nominal distance. This shows that the human body does not permanently degrade the ranging link, but can introduce large short-duration errors.

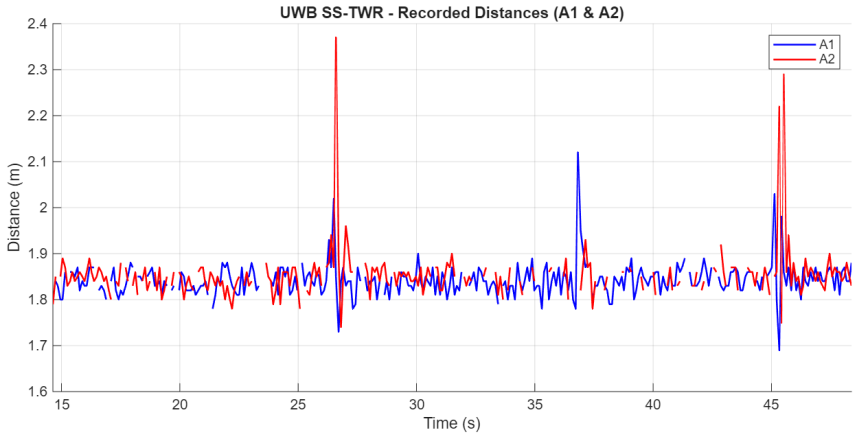


FIGURE 1.14
UWB distance measurements during the human-body dynamic obstruction test. The main peaks occur around 26 s, 37 s, and 45 s, corresponding to the moments when the person was placed between the tag and the anchors.

This behavior confirms that the human body can strongly affect UWB ranging when it crosses or obstructs the propagation path. Compared with the plastic-board tests, the degradation is more localized in time but has a larger amplitude: the signal is not continuously degraded, but sharp peaks appear when the person is between the devices. This is representative of a real Follow-me scenario, where the user or another person may temporarily block the direct path during motion.

For the control system, this result is important because a single raw distance sample may become temporarily unreliable during a human-body obstruction. The controller should therefore not react directly to isolated peaks. Instead, the ranging data should be filtered and monitored in order to reject short outliers, handle temporary dropouts, and preserve stable robot motion.

METALLIC OBSTRUCTION TEST

A second series of attenuation tests was carried out using a laptop computer as an obstructing object. In contrast with the plastic board, the laptop represents a more challenging obstacle because it contains a large metallic surface that can strongly affect UWB propagation. Such an object is representative of realistic indoor environments, where metallic items such as computers, cabinets, or equipment may partially block or reflect the signal path.

Three configurations were investigated. In the first one, the laptop was placed in front of the anchors. In

the second one, it was placed in front of the tag. In the third one, the laptop was intentionally introduced and removed during the acquisition in order to observe the transient effect of the sudden appearance of a metallic obstacle on the ranging measurements.

LAPTOP PLACED IN FRONT OF THE ANCHORS In the first configuration, the laptop was positioned between the tag and the anchors, close to the anchor side. The corresponding setup is shown in Figure 1.15. The objective of this test was to determine how a metallic obstacle located near the receiving devices affects the measured distances.

(a) Laptop in front of the anchors

(b) Laptop in front of the tag

FIGURE 1.15
Experimental configurations used for the metallic obstruction test. The laptop was successively placed in front of the anchors and in front of the tag while keeping the reference distance fixed at approximately 1.85 m.

The corresponding ranging measurements are presented in Figure 1.16. This configuration produces one of the strongest degradations observed in the obstacle tests. Compared with the reference line-of-sight case and the plastic-board case, the measurements show a clear increase in variability and a significant number of missing samples, especially on Anchor A1. The discontinuities in the A1 curve indicate that the ranging exchange was frequently interrupted or invalidated when the laptop was placed close to the anchors.

In addition to these timeouts, both anchors show a positive shift of the measured distance compared with the nominal value of approximately 1.85 m. Anchor A1 is particularly affected, with large fluctuations and peaks above 2.2 m, while Anchor A2 remains more continuous but is still biased around 2.0 m. This suggests that the laptop does not only add random noise, but also modifies the propagation path through attenuation, reflection, and multipath effects.

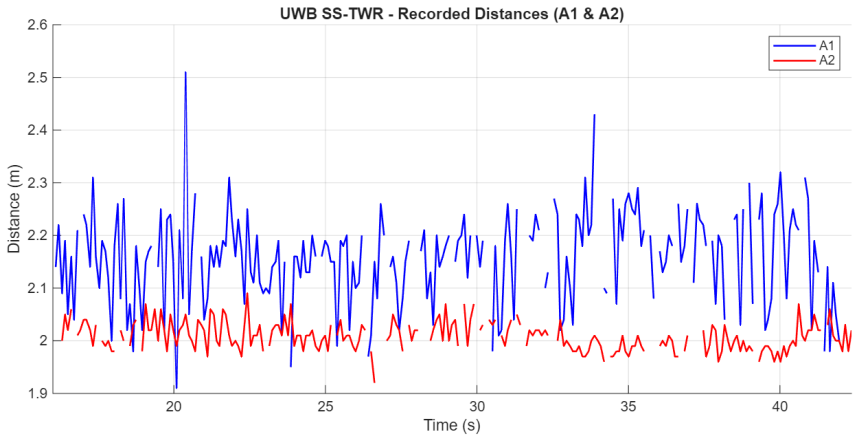


FIGURE 1.16
UWB distance measurements with the laptop placed in front of the anchors. This configuration produces strong fluctuations, a positive distance bias, and several missing samples, especially on Anchor A1.

LAPTOP PLACED IN FRONT OF THE TAG In the second configuration, the same laptop was placed close to the tag side, again between the devices. This setup makes it possible to evaluate whether the location of the metallic obstacle influences the ranging behavior differently when the obstruction is near the transmitting device rather than near the anchors.

The measured distances for this configuration are shown in Figure 1.17. Compared with the laptop placed in front of the anchors, the measurements are more continuous, but several strong outliers appear, especially on Anchor A2. Most of the acquisition remains close to the nominal distance range, but short peaks above 2.0 m are observed after approximately 25 s. A physically unrealistic negative value also appears around 31 s on Anchor A2.

This negative value cannot be interpreted as a valid distance measurement. It is therefore considered as an invalid outlier, most likely caused by a corrupted ranging exchange or by an erroneous time-of-flight estimation during the obstruction. This test shows that placing the laptop close to the tag does not necessarily produce as many timeouts as the anchor-side configuration, but it can generate severe isolated errors. These errors are particularly important for the control system, since reacting to a single invalid sample could lead to an incorrect robot command.

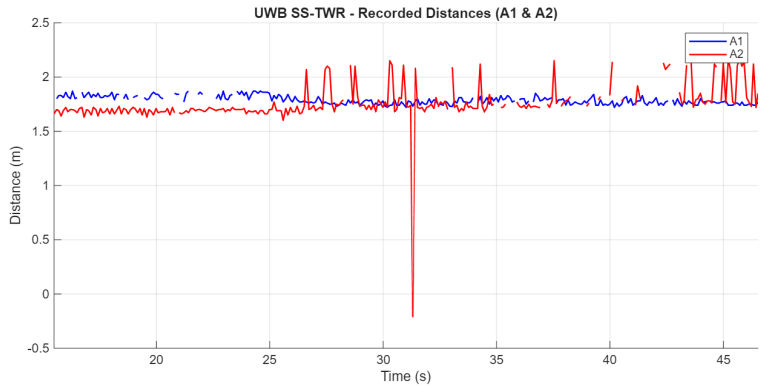


FIGURE 1.17
UWB distance measurements with the laptop placed in front of the tag. The signal remains mostly measurable, but several strong outliers appear, including one physically invalid negative value on Anchor A2.

DYNAMIC APPEARANCE OF THE LAPTOP Finally, a dynamic obstruction test was performed by introducing the laptop into the propagation path and then removing it during the measurement. This experiment is representative of realistic situations in which a metallic object may temporarily block the line of sight between the tag and the anchors.

The recorded distances are shown in Figure 1.18. In this test, the laptop was introduced into the propagation path during two distinct time intervals, approximately from 25 s to 30 s and from 35 s to 40 s. These two intervals are clearly visible in the measurements. Before the first obstruction, the distances remain close to the nominal range. When the laptop is introduced, both anchors exhibit a sudden increase in the measured distance, with values reaching approximately 2.3 m to 2.5 m.

After the laptop is removed, the measurements rapidly return close to the nominal distance. The same behavior is observed during the second obstruction interval between approximately 35 s and 40 s. This before-during-after behavior provides the clearest evidence that the metallic object is responsible for the observed degradation. The laptop causes a temporary but strong increase in distance error and measurement variability, while the ranging system recovers once the obstacle is removed.

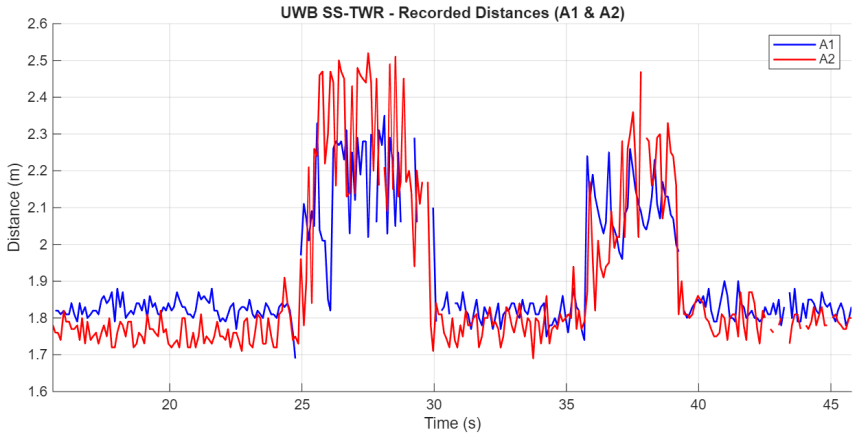


FIGURE 1.18
UWB distance measurements during the dynamic laptop obstruction test. The laptop was introduced between approximately 25–30 s and 35–40 s, producing large temporary distance errors before the measurements returned close to the nominal range after removal.

Overall, the laptop tests show that metallic-component obstacles have a much stronger influence on UWB ranging than the plastic board. Three different degradation mechanisms were observed. First, when the laptop was placed in front of the anchors, the measurements became discontinuous and several timeouts appeared, especially on Anchor A1. Second, when the laptop was placed close to the tag, the signal remained more continuous but severe isolated outliers were observed. Third, when the laptop was dynamically introduced and removed, the measured distances increased strongly during the obstruction intervals and returned close to the nominal value after removal.

These results show that a metallic object can affect the UWB system in several ways: it can increase the measured distance bias, generate missing samples, and create large outliers. For a Follow-me robotic application, this is a critical observation. The controller should not use raw distance samples directly, especially in environments containing metallic objects. Filtering, dropout handling, and outlier rejection are therefore required to obtain a robust distance estimate before generating motion commands.

PART 2

SIMULATION

2.1 WHAT WE WANTED TO DO

Part I of this project focused on getting the UWB distance measurements right. However, a distance measurement alone is not sufficient to make a robot follow a user. To actually build a *Follow-me* rover, we need to put those distances into a full system: a robot that reads them, figures out where the user is, and drives itself toward the user while keeping a safe distance.

Building that whole system directly on real hardware is slow and risky. A bug in the code can easily turn into a robot crashing into a wall. So we decided to first *simulate* everything: build the two rovers inside a virtual 3D world, give them virtual UWB sensors, and make the simulated follower chase the simulated leader using exactly the same code that will eventually run on the real hardware. Once it works in simulation, we plug in the real sensors and the real motors, and the code does not change.

The simulation answers three practical questions:

- Does our follow-me logic work at all when the leader moves around?
- How does it behave when the UWB measurements are noisy (because real sensors are never perfect)?
- What extra information does the robot need beyond the two distances? (Spoiler: a gyroscope.)

2.2 HOW THE SIMULATION IS BUILT

We used two open-source tools that are standard in robotics research:

Gazebo Harmonic is a physics simulator. It knows about gravity, friction, wheel rotation, and collisions. We describe each rover as a simple 3D model (a box for the body, two cylinders for the wheels, small cubes on top for the UWB anchors and for the UWB tag), and Gazebo handles the physics of how the rover moves when we command it.

ROS 2 (Robot Operating System) is the "glue" that connects all the pieces together. It is built on the idea of *topics*: different pieces of code run as independent programs, and they exchange data by publishing messages on named channels. For example, one program publishes the UWB distances on a topic called `/uwb/anchor1/range`, and another program listens to that topic to compute the tag position. The programs don't need to know anything about each other — they just agree on the topic name and the message format. This is what lets us swap the simulated UWB sensor for the real one later on without touching any other code.

2.2.1 THE TWO SIMULATED ROVERS

The simulation contains two rovers:

- An **orange leader** carrying the UWB tag (a green pillar on top).
- A **blue follower** carrying the two UWB anchors (the cyan cubes on top).

Both have a small white "nose" pointing forward so we can tell their orientation at a glance. Figure 2.1 shows the two rovers at the start of a run.

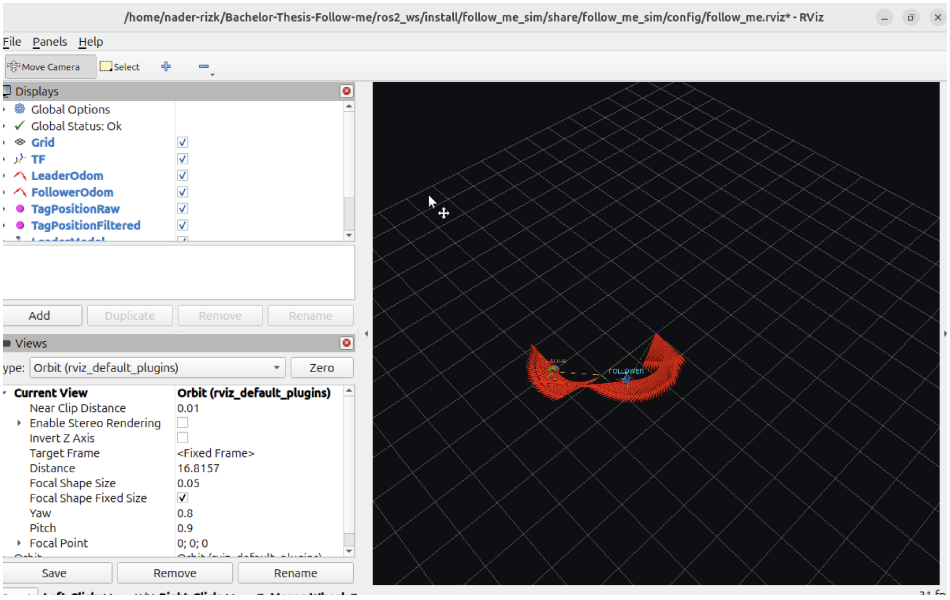


FIGURE 2.1
Starting configuration in RViz. The orange rover is the leader (with the green tag), the blue rover is the follower (with the two cyan anchors). The small white blocks on top indicate the forward direction of each rover.

2.2.2 THE PIPELINE

When the simulation runs, the following happens every few hundred milliseconds:

1. The fake UWB sensor computes the true distance from the leader's tag to each of the follower's two anchors. Then it adds realistic noise on top (more on that in Section 2.4).
2. The *localization* program reads the two distances and computes a position for the tag relative to the follower (roughly: "the leader is 2 m ahead and 0.5 m to my left").
3. A *filter* smooths out the noise, so small jitter in the measurements does not translate into jerky robot motion.
4. A *controller* compares where the tag is to where the follower wants it to be (at a set distance of 1 m), and commands the follower's wheels to close the gap.

The controller output is a command like "drive forward at 0.3 m/s while turning left at 0.5 rad/s," which Gazebo then applies to the simulated wheels. This forms the closed-loop control pipeline used in the simulation.

2.3 THE TWO-ANCHOR PROBLEM (AND WHY WE NEED A GYROSCOPE)

Here is the subtle part. When you have only *two* UWB anchors on the follower, the two distances you measure are not enough to fully locate the tag. There is an ambiguity: the tag could be in *front* of the follower, or in a *mirror position behind* the follower, and both positions would give the same pair of distances. The two distance measurements alone are therefore insufficient to distinguish between these two symmetric solutions.

Figure 2.2 illustrates what happens during a simulation run. The follower (blue, bottom-right) is chasing the leader (orange, top-left). The green line is its recent path. The red dots are the raw tag positions estimated from the UWB distances. You can see two groups of red dots: one near the real leader (the correct answer), and one in the opposite direction (the mirror answer). Without help, the follower does not know which group to trust.

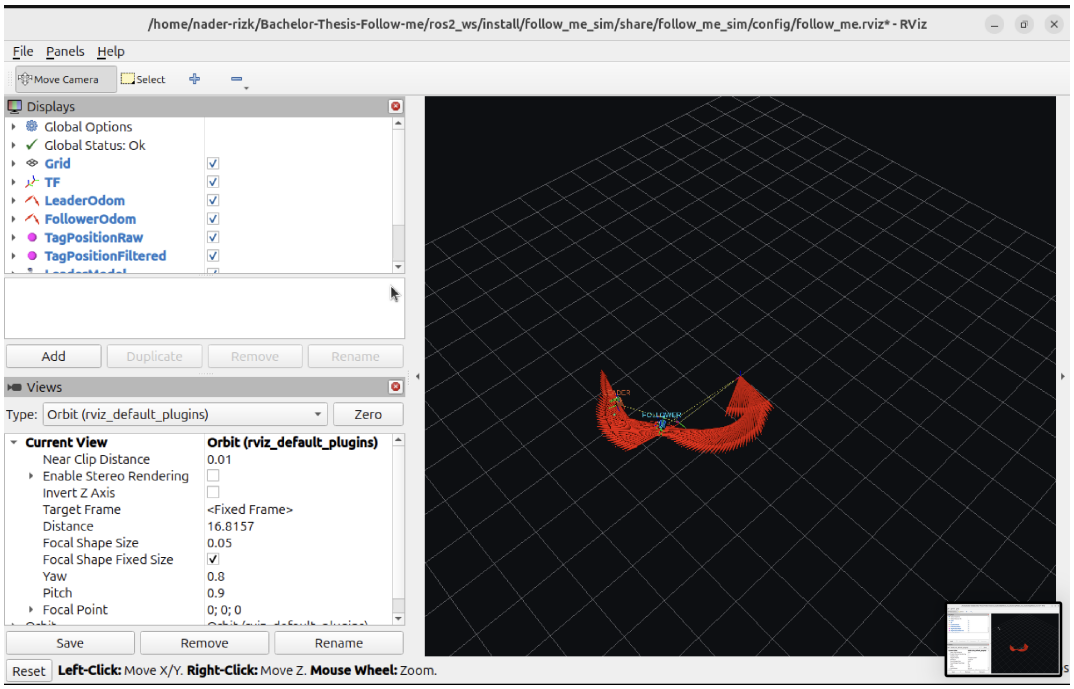


FIGURE 2.2

A simulation run in RViz. The blue follower is chasing the orange leader. The red dots are the raw estimated positions of the tag; notice there are two clusters, symmetric with respect to the follower, because of the two-anchor ambiguity. The gyroscope-based logic picks the correct cluster.

This is where the **gyroscope** comes in. A gyroscope tells the follower how fast it is rotating. When the follower turns, the *real* tag position barely moves in the world (the leader did not teleport), but the *mirror* position swings wildly because it is a geometric reflection that moves with the follower. By comparing how much each candidate position moves when the follower rotates, we can detect which one is real and which one is the mirror.

Practically, the follower spins in place for about 2.5 seconds at the beginning of each run. This small rotation is enough to reveal which of the two candidates is the real tag. Once that is established, the follower stops spinning and starts actually chasing the leader.

We chose a gyroscope specifically because it is the cheapest and smallest sensor that does this job well. We are not using the full Inertial Measurement Unit (accelerometer, magnetometer, etc.) — only the gyroscope part that measures rotation rate.

2.4 TESTING THE SIMULATION WITH REAL DATA

A simulation is only as useful as its realism. If the simulated UWB sensor was perfect (no noise, no glitches), the software would work flawlessly in simulation but fail on the real rover. So we ran a measurement campaign on the real DW3000 bench, analysed the data, and injected the same kind of noise into the simulation.

2.4.1 STATIC AND DYNAMIC MEASUREMENTS AT KNOWN DISTANCES

We put the tag at a measured distance from the anchors and let the system record for about a minute. Figures 2.3 and 2.4 show two of those recordings. In each plot:

- The blue line is the distance reported by anchor 1.
- The red line is the distance reported by anchor 2.
- The horizontal axis is time.

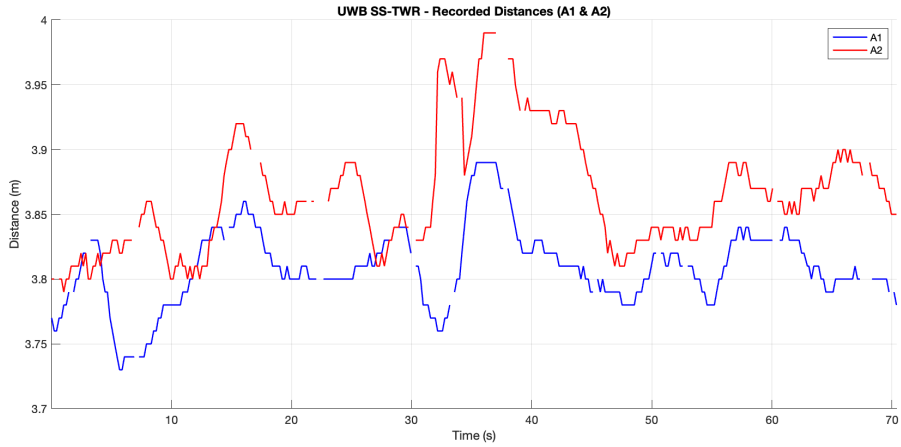


FIGURE 2.3
Static measurement at around 4 m. The two anchors stay near 3.8 m and 3.9 m. The random fluctuations are slightly larger than at 1 m, which is expected: UWB noise grows with distance.

From these static tests we extracted, for each anchor:

- The **average error** (how far off from the true distance the sensor is on average — the "bias").
- The **random noise** (how much the measurement jumps around from sample to sample).
- The **dropout rate** (how often a measurement is lost, roughly 5% in our bench after the timeout tuning from Part I).
- Occasional **multipath events** where one anchor temporarily reports a distance that is 30–70 cm too large for a few seconds, because the signal bounced off something before reaching the antenna. This happened only on anchor 2 in our setup.

To make sure the system also works when the tag is moving, we recorded a run where the tag was carried by hand: it started far away, came close to the anchors, then moved away again. Figure 2.5 shows the result.

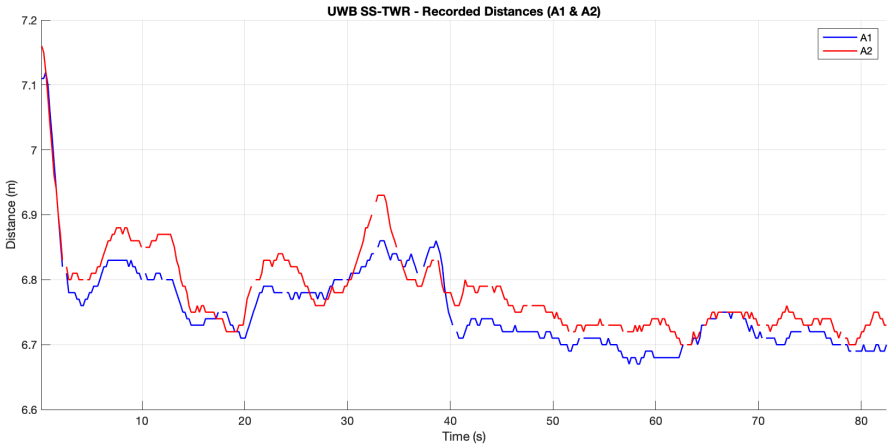


FIGURE 2.4
Static measurement at around 7 m. Still stable overall, but the short transient at the very beginning shows that the system takes a second or two to settle once it starts ranging.

2.4.2 PLUGGING THE REAL NOISE INTO THE SIMULATION

We wrote a small analysis tool that reads the recorded CSV files and prints all the noise statistics automatically. We then copied those numbers into the simulation configuration. The simulated UWB sensor now adds the same kind of noise, the same bias, the same occasional multipath glitches, and the same dropout rate as the real one.

This means that if the software works in simulation now, it has already been tested under realistic noise. This reduces the uncertainty associated with the transition to the real rover.

2.5 WATCHING THE SIMULATION RUN

Figure 2.6 shows a later moment in a full simulation run. After the startup spin, the follower locks onto the correct branch (the real leader position, not the mirror) and starts chasing. The red dots stay close to the leader, the follower's green path trails the orange leader's path, and the overall distance stays around 1 m. In the terminal, the localization program prints a steady message saying which branch it has chosen (e.g. `sign=+1`) and how confident it is.

When the leader turns, the follower turns too. When the leader slows down, the follower keeps its distance. When one of the simulated UWB anchors suddenly reports a 50 cm jump because of simulated multipath, the filter absorbs it and the follower keeps going without jerking around.

2.6 LIMITATIONS

The simulation is useful, but several simplifications remain compared with the real system. Several things are simpler in simulation than they will be on the real rover:

- **The wheels do not slip in simulation.** In real life, especially on carpet or a slippery floor, wheel odometry drifts much faster than what we see in simulation. This is one reason we are using the gyroscope: to get a cleaner orientation that does not depend on the wheels.
- **The gyroscope in simulation is too clean.** Real gyroscopes have a slow drift in their bias (the offset drifts a bit over minutes). Our simulated one does not. On the real rover we will need to

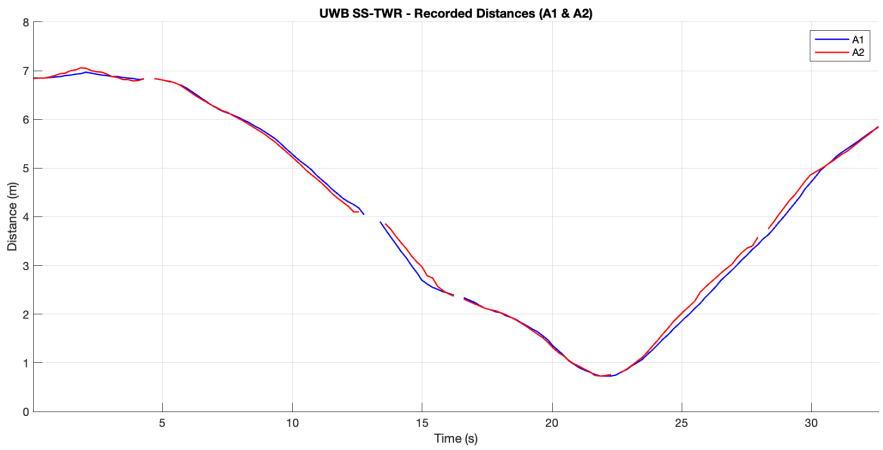


FIGURE 2.5
Dynamic measurement. The tag approaches the anchors from about 7 m, reaches a minimum around 0.7 m, then moves back. The blue and red lines track the same trajectory very closely, which confirms the system reacts correctly to motion.

briefly calibrate the gyro at the start of each session.

- **Multipath events are random in simulation.** On the real hardware, multipath depends on the geometry of the room (walls, people, furniture). The simulation injects random multipath events at a rate we measured on the bench, but the actual pattern in a specific real environment can differ.
- **Obstacle avoidance is not included in the current simulation.** The simulation world is an empty flat plane. We have not tested obstacle avoidance because that is outside the scope of this project.

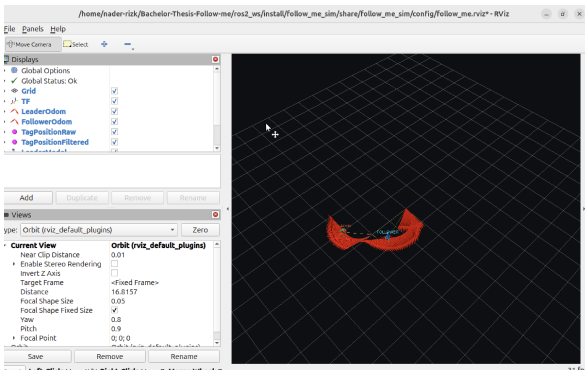
None of these limitations invalidates the simulation — they just mean that the real hardware will require a bit of tuning, especially on the gyroscope calibration side. But the core algorithms are validated.

2.7 BOTTOM LINE

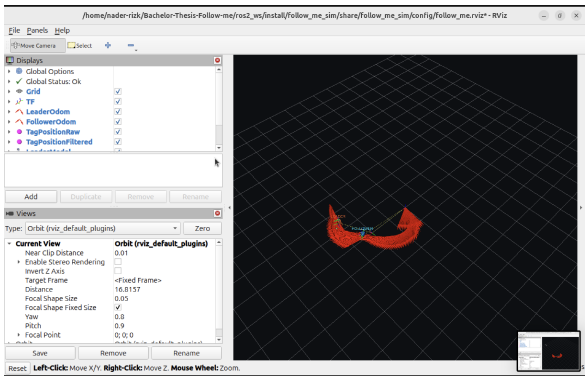
The simulation successfully demonstrates that:

1. A rover can follow another rover using only two UWB anchors and a gyroscope.
2. The mirror ambiguity of the two-anchor setup is resolved in practice by a short initial spin.
3. Realistic UWB noise does not break the system, because it was calibrated against real bench data.
4. The code is structured so that the transition to the real rover is a matter of swapping the simulated sensor nodes for real driver nodes. No algorithm has to be rewritten.

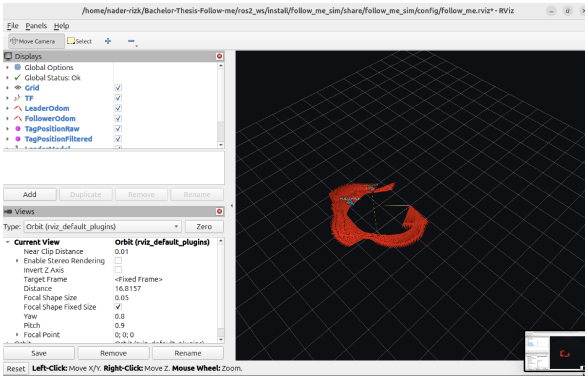
The next step was therefore to transfer this validated architecture to the physical rover, as described in Part III.



(A) Startup: both rovers spawned, follower about to begin the initial spin.



(B) During the spin: both hypothesis clusters (red dots) are still visible before the gyroscope resolves the ambiguity.



(C) Chase engaged: the follower has locked onto the correct branch and is closing on the leader.

FIGURE 2.6
Three snapshots of a full simulation run, from startup (top-left) to steady-state following (bottom-right).

PART 3

SIMULATION APPLICATION ON AN ACTUAL ROBOT PLATFORM

3.1 INTRODUCTION

After validating the full follow-me pipeline in simulation (Part II), the next logical step was to move the same software stack onto a real physical platform. The objective of this third part of the project is therefore to demonstrate that the architecture developed during simulation is portable to a real robot, and that no part of the localization, filtering, or control logic has to be rewritten in the transition.

The physical platform used for the experiments is a four-wheel mecanum rover. The rover runs an embedded ESP32 firmware that exposes a small HTTP API, controls its motors through per-wheel PID loops, and uses a BNO08x IMU for internal heading estimation. Building on top of an existing, validated robot platform avoided spending time on mechanical design and motor calibration, and allowed the focus to remain on the integration of the UWB-based control loop.

3.2 MOUNTING THE UWB ANCHORS

The two UWB anchors are mounted on a plate, along the lateral axis, symmetrically with respect to the centre of the chassis. The relevant design parameter for the trilateration code in ROS is the *baseline*, that is the distance between the antennas of the two anchor boards. We did not place them on the rover directly as this needed external power sources that we did not have time to put in place

```
localization:
  ros__parameters:
    baseline: 0.27
```

Both anchors are mounted with their patch antennas pointing upward, away from the metal. This orientation reduces multipath reflections off the table rover itself, which were observed during the static bench measurements presented in Part I. The boards are fixed with double-sided VHB foam tape; this proved sufficient to keep them rigid during all of the tests, including sharp turns.

3.3 EMBEDDED FIRMWARE

The embedded firmware running on the rover implements several features that we did not need to redevelop:

FIGURE 3.1
Tag mounted on the rover, with anchors placed as described above

- Per-wheel PID velocity control, with gains tunable through a small web interface on the rover itself.
- Reading of the four encoders and computation of the body-frame velocity using the standard mecanum inverse-kinematic equations.
- Reading of the BNO08x IMU and computation of a yaw estimate using the fused game-rotation-vector quaternion.
- A small embedded web server exposing the live telemetry, a virtual D-pad for manual control, and a set of hard-coded test paths.
- A WiFi access point so that any device on the same network can reach the rover at the fixed IP address `192.168.4.1`.

For the follow-me application, the only firmware feature that the ROS pipeline interacts with is the HTTP endpoint:

```
GET http://192.168.4.1/move?vx=<mm/s>&vy=<mm/s>&w=<deg/s>
```

When this endpoint receives a request, the firmware switches out of any active path-following mode, parses the three velocity values, and feeds them as setpoints into its per-wheel PID loops. The rover then holds those setpoints until the next request arrives. The firmware does *not* implement a timeout on this endpoint, which means that the rover would keep moving indefinitely if the ROS bridge stopped sending commands. A watchdog mechanism on the ROS side, described in the next section, was therefore added to compensate for this.

The WiFi credentials of the access point were customised for our tests to:

```
const char* kSsid = "<network_name>";
const char* kPassword = "<password>";
```

This is the only modification made to the firmware: the network name and password were changed so that the laptop running ROS could be connected to a known, dedicated wireless network during the experiments. All other firmware parameters, including the geometry constants, the PID gains, and the maximum allowed wheel speed, were left at the values calibrated by the Mecanum Pro team.

3.4 ROS BRIDGE

The role of the ROS bridge is to translate the velocity commands produced by the follow-me controller (developed in Part II) into HTTP requests that the firmware understands. The bridge is implemented as a small standalone Python node, `http_bridge_node.py`, located in the `test/` folder of the project repository. It is not packaged as a regular ROS node because it is intentionally kept outside of the `follow_me_uwb` package as long as the integration is not fully validated.

3.4.1 SUBSCRIPTIONS AND CONVERSIONS

The bridge subscribes to `/follower/cmd_vel`, a `geometry_msgs/Twist` topic published by the follow-me controller node. The Twist message uses the ROS convention REP-103, where `linear.x` is the forward velocity in metres per second, `linear.y` is the lateral velocity (positive to the left) in metres per second, and `angular.z` is the yaw rate in radians per second.

The firmware, on the other hand, expects velocities in millimetres per second and angular rates in degrees per second. Each tick of the bridge therefore performs the following conversion:

```
vx_mm = msg.linear.x  * 1000.0          // m/s   -> mm/s
vy_mm = msg.linear.y  * 1000.0          // m/s   -> mm/s
w_deg = msg.angular.z * (180.0 / PI)    // rad/s -> deg/s
```

The three values are then clamped to the safe operating range configured by the parameters `vmax_mms` and `wmax_dps`, and serialised as query parameters in the URL of the HTTP request.

3.4.2 PERIODIC POSTING AND WATCHDOG

A timer running at 20 Hz posts the most recent command to the firmware. Each request is sent using a `requests.Session()` object so that the underlying TCP connection is reused between calls, which reduces latency and avoids exhausting the limited number of sockets that the embedded server can handle. The HTTP timeout is set to 50 ms, which is short enough that a temporarily unreachable rover does not block the ROS executor.

To compensate for the absence of a watchdog on the firmware side, the bridge tracks the timestamp of the last received `cmd_vel` message. If no new command arrives within `cmd_timeout_s` (default 0.5 s), the periodic timer automatically switches to sending zero-velocity requests. As a result, if the ROS controller crashes or if the WiFi link goes down, the rover comes to a stop within half a second instead of continuing on its last setpoint.

3.4.3 LAUNCH FILE

A single launch file, `launch_hardware.launch.py`, orchestrates the full stack for the physical test. It starts the UWB driver, the localization node, the filter, the controller, the synthesised odometry node, the HTTP bridge, and a static transform broadcaster. A `bridge_only` flag allows starting only the bridge and the synthesised odometry node, which is useful for the early integration tests where the UWB hardware is not yet connected:

```
ros2 launch test/launch_hardware.launch.py \
    bridge_only:=true esp_ip:=192.168.4.1
```

The full pipeline is started by:

```
ros2 launch test/launch_hardware.launch.py \
    serial_port:=/dev/ttyUSB0 esp_ip:=192.168.4.1
```

3.5 PRACTICAL ISSUES AND RESOLUTIONS

The transition from simulation to physical hardware revealed several practical issues that were not visible during the simulation work. Each of them is documented below for future reference.

3.5.1 ARDUINO LIBRARY INCOMPATIBILITY

The first issue encountered was a compilation error in the `ESPAsyncWebServer` library. The error referred to functions such as `mbedtls_md5_starts_ret`, which no longer exist in the mbedtls API shipped with the recent ESP32 Arduino core (version 3.x). The originally maintained version of the library by `me-no-dev` has not been updated to follow this API change.

The fix consisted in replacing the obsolete library by its actively maintained fork from the `ESP32Async` organisation:

```
cd ~/Documents/Arduino/libraries
rm -rf ESPAsyncWebServer AsyncTCP
git clone https://github.com/ESP32Async/AsyncTCP.git
git clone https://github.com/ESP32Async/ESPAsyncWebServer.git
```

After this change, the firmware compiled cleanly with the latest ESP32 core.

3.5.2 UPLOAD FAILURES AT HIGH BAUD RATE

The first flash attempts failed during the upload step. The board was detected correctly at the initial 115,200 bps rate, but the upload crashed as soon as `esptool` switched to the higher 921,600 bps rate normally used for the actual writing of the firmware. The failure was systematic across two different Adafruit Feather ESP32 V2 boards and on different USB cables, which suggests an instability of the auto-baud sequence on this particular combination of USB-serial chip and host driver on macOS.

The fix was to force a lower upload speed in Arduino IDE:

```
Tools -> Upload Speed -> 115200
```

At this lower rate, the complete flash takes about one minute, but the process is fully reliable. This setting is preserved on the firmware side and does not affect the rover's runtime behaviour.

3.5.3 MANUAL ENTRY INTO THE BOOTLOADER

Even at 115,200 bps, the upload was sometimes interrupted by the error `Failed to connect to ESP32: No ser` This indicates that the chip did not enter the boot ROM at the moment `esptool` started sending the synchronisation sequence.

The robust workaround is to manually force the bootloader before the upload begins. With the Feather V2, this is done by holding the `BOOT` (sometimes labelled `IO0`) button, briefly pressing and releasing the `RESET` (or `EN`) button, and then releasing `BOOT`. The Feather is then in download mode and `esptool` can flash it without depending on the DTR/RTS auto-reset sequence.

3.5.4 DEFECTIVE WHEEL ON THE ORIGINAL BOARD

During the first hardware test, one of the four wheels of the rover did not turn while the other three responded normally to direction commands. The faulty wheel corresponded to encoder pins routed on GPIO 34 and GPIO 39, which are *input-only* pins on the ESP32-PICO without internal pull-up or pull-down resistors. The boot log confirmed this with the warning:

```
E (gpio) gpio_pullup_en : GPIO number error (input-only pad has no internal PU)
```

Without external pull resistors, the encoder signals on these pins floated and the PID never received a valid feedback for that wheel. The rover behaved as if the wheel was mechanically blocked: the PID saturated its integral term and the chassis rotated around the dead wheel instead of moving in a straight line.

The fix was to swap the original Feather V2 board for a second unit that had the encoders correctly wired to GPIOs supporting internal pull-ups. After flashing the same firmware on the new board, all four wheels responded as expected to the on-board web interface.

3.5.5 WIRELESS CONNECTIVITY

A minor but recurrent issue was that the laptop running ROS needed to be connected to the same WiFi network as the rover during the experiments. Since the rover acts as an access point with no internet gateway, the laptop loses internet access for the duration of the test. All Python dependencies of the ROS bridge therefore had to be installed beforehand:

```
sudo apt install python3-requests
```

This is a one-time operation. Once installed, the bridge runs entirely on local network traffic and does not require internet access during the test.

3.6 FINAL INTEGRATION TEST

After resolving the issues above, the full follow-me pipeline was launched on the physical rover. The procedure followed for the test is the one documented in the project repository under `test/checklist.md` and is summarised below.

1. The rover is powered on and creates the WiFi access point `Nader`.
2. The laptop is connected to `Nader` and the connection is verified by pinging `192.168.4.1`.
3. A first manual test is performed with `curl` in order to confirm that the rover obeys the velocity endpoint and that the watchdog correctly stops the wheels once the requests stop.
4. The bridge-only configuration of the launch file is started. Manual `ros2 topic pub` commands are sent on `/follower/cmd_vel` to check that the conversion to HTTP requests is correct and that the rover responds.
5. The UWB hub is connected to the laptop over USB. The full launch file is started with the appropriate serial port and the rover's IP address.
6. The UWB tag is placed approximately two metres in front of the rover, and the follow-me behaviour is engaged.

3.7 CONCLUSION

The objective of this third part was to verify that the architecture developed and validated in simulation could be transferred to a real robotic platform without requiring modifications to the core localization and control algorithms. The integration process demonstrated that this objective was largely achieved.

The physical rover was successfully connected to the ROS-based follow-me framework through the HTTP bridge, allowing velocity commands generated by the controller to be transmitted to the embedded firmware in real time. The various practical issues encountered during deployment, including Arduino library

incompatibilities, firmware upload difficulties, hardware faults, and wireless connectivity constraints, were identified, documented, and resolved.

Most importantly, the transition from simulation to hardware confirmed the modular design of the software architecture. The localization, filtering, and control components developed in Part II remained unchanged, while only the simulated interfaces were replaced by their physical counterparts. This validates one of the key design goals of the project: separating the high-level follow-me logic from the underlying hardware implementation.

Although the complete autonomous follow-me demonstration remains the next step of the project, the work presented here establishes all the necessary foundations. The UWB ranging system has been characterized and calibrated, the localization and control algorithms have been validated in simulation under realistic conditions, and the communication pipeline to a real rover has been successfully integrated. The project is therefore well positioned for the final phase, which consists of deploying the UWB sensors on the rover and evaluating the complete follow-me behavior in real-world conditions.

PART 4

CONCLUSION AND ACKNOWLEDGMENTS

GENERAL CONCLUSION

The objective of this Bachelor project was to develop and validate the main building blocks of a *Follow-me* robotic system based on Ultra-Wideband ranging. The project was structured around three complementary stages: the experimental characterization of the UWB modules, the development of a simulation environment, and the first integration of the architecture on a real robotic platform.

The first part focused on implementing a UWB ranging system using DW3000 transceivers and ESP32 microcontrollers. The setup was progressively extended from one tag and one anchor to a two-anchor configuration. This allowed us to study the influence of timing parameters, antenna delays, timeout values, module orientation, and obstacles on distance measurements. The experiments showed that UWB can provide stable and accurate estimates, but also that it remains sensitive to calibration, propagation conditions, and obstructed line-of-sight situations. Metallic objects and human-body obstructions, in particular, introduced outliers, bias, and occasional missing measurements, confirming the need for filtering and robust data handling.

The second part addressed these limitations through simulation. A complete follow-me pipeline was implemented in Gazebo and ROS 2, including simulated UWB measurements, localization, filtering, and control. The simulation validated the overall architecture before hardware deployment and highlighted the two-anchor ambiguity problem, where the estimated tag position can be mirrored with respect to the follower. A gyroscope-based initialization was introduced to resolve this ambiguity. By injecting noise and disturbances based on real UWB data, the simulation provided a realistic validation environment.

The third part focused on transferring the simulated architecture to an actual mecanum rover. A ROS-to-HTTP bridge was developed to convert controller velocity commands into commands understood by the rover firmware. Several practical issues were encountered and resolved, including Arduino library incompatibilities, upload failures, bootloader access, hardware faults, and wireless connectivity constraints. This confirmed the modularity of the architecture: the localization, filtering, and control algorithms developed in simulation could be reused, while only the hardware interface layer had to be adapted.

Overall, the project established the foundations of a UWB-based *Follow-me* robotic system. The ranging system was experimentally characterized, the localization and control pipeline was validated in simulation, and the first connection to a real robotic platform was achieved. While the final fully autonomous demonstration still requires further work, the main technical components are now in place.

AREAS OF IMPROVEMENT AND OPTIMIZATION

Several improvements could be made to increase the robustness, accuracy, and practical usability of the system.

First, the UWB calibration could be further refined. The antenna delays were tuned experimentally, but a more systematic calibration procedure could reduce the remaining distance bias between anchors. Each UWB board could be calibrated individually, since small hardware differences between modules were observed during the experiments. A more complete calibration campaign over different distances, orientations, and environments would make the system more reliable.

Second, the filtering strategy could be improved. The current approach is sufficient to smooth measurement noise, but stronger outlier rejection would be useful in the presence of human-body obstruction, metallic objects, or multipath effects. For example, a Kalman filter, an extended Kalman filter, or a particle filter could combine UWB measurements with odometry and gyroscope data. This would allow the system to reject isolated unrealistic values and maintain a stable estimate of the user position even during short measurement dropouts.

Third, the positioning system could be extended beyond two anchors. The two-anchor setup is simple and compact, but it introduces a geometric ambiguity and provides limited information. Adding a third anchor on the rover would remove the mirror ambiguity and improve localization accuracy. A larger anchor baseline could also improve angular resolution, although this would need to be balanced with the physical constraints of the robot.

Another important improvement would be the integration of additional perception sensors. A LiDAR could be used to detect obstacles around the robot and prevent collisions during follow-me operation. This would be especially important in indoor environments where furniture, walls, or people may appear between the follower and the user. Computer vision could also be added, for example using a camera to detect and track the user visually. Combining UWB with vision would make the system more robust: UWB provides reliable distance information even when visual tracking is difficult, while vision can help identify the correct target and improve direction estimation.

The control strategy could also be optimized. The current controller is designed to validate the follow-me behavior, but more advanced control laws could improve smoothness and safety. Speed limits, acceleration limits, obstacle-aware trajectory planning, and emergency stop conditions should be added before deploying the system in a real environment with people. The controller should also handle temporary loss of UWB measurements by slowing down or stopping the robot rather than continuing blindly.

Finally, the hardware integration should be made more compact and robust. The UWB anchors currently require careful placement and external wiring. A dedicated mounting structure, reliable power supply, and protected electronics would make the system easier to use and more repeatable during experiments.

REMAINING STEPS

The next step is to complete the deployment of the full follow-me system on the physical rover. This requires mounting the UWB anchors permanently on the robot, placing the tag on the user, and running the complete ROS pipeline with real UWB measurements instead of simulated ones.

Once this is achieved, the system should be tested in simple scenarios: first with the user standing still at different positions, then walking slowly in a straight line, and finally moving along more complex trajectories. These tests would make it possible to compare the real behavior of the rover with the simulation results and to tune the controller gains accordingly.

After this initial validation, robustness tests should be performed in more realistic indoor conditions. The system should be evaluated with partial obstructions, different orientations of the user, nearby metallic objects, and other people moving in the environment. These tests would help determine the limits of the UWB-only approach and identify when additional sensors become necessary.

The integration of a LiDAR would then be a natural next step. It would allow the robot to detect obstacles and avoid collisions while following the user. In parallel, computer vision could be explored to recognize and track the target person. A future version of the system could therefore combine UWB, gyroscope data, odometry, LiDAR, and camera information into a single sensor-fusion framework.

The final objective would be a complete autonomous demonstration in which the rover follows a user smoothly and safely in an indoor environment, maintains a desired distance, avoids obstacles, and remains robust to temporary UWB disturbances. This would represent the full realization of the *Follow-me* concept introduced at the beginning of the project.

ACKNOWLEDGEMENTS

We would like to thank Professor Christophe Salzmann for his supervision, guidance, and support throughout this Bachelor project. His advice helped us define the objectives of the project, overcome technical difficulties, and progressively improve the quality of our work.

We would also like to thank the Automatic Control Laboratory at EPFL for providing the environment, equipment, and resources necessary to carry out this project. The support of the laboratory made it possible to experiment with UWB modules, simulation tools, and robotic platforms, and greatly contributed to the development of this work.

BIBLIOGRAPHY

[1] Qorvo. *DW3110 Ultra-Wideband Transceiver*.
https://www.qorvo.com/products/p/DW3110.

[2] Qorvo. *DW3000 Datasheet*.
https://www.mouser.com/datasheet/2/412/Qorvo_DW3000-2934245.pdf.

[3] Qorvo. *DWM3000 Data Sheet*.
https://www.mouser.com/datasheet/2/412/DWM3000_Data_Sheet-2932971.pdf.

[4] Qorvo. *DWM3000 Module Documentation*.
https://www.qorvo.com/products/d/da008334.

[5] Qorvo. *Qorvo UWB Technical Document*.
https://forum.qorvo.com/uploads/short-url/xD3TlXKvkujjdUaJWXv2b4E7GQN.pdf.

[6] Qorvo. *Qorvo UWB Technical Document*.
https://forum.qorvo.com/uploads/short-url/oW6ojIIo7adaWf6S7NnKzyN7OSi.pdf.

[7] Qorvo. *Qorvo UWB Technical Document*.
https://forum.qorvo.com/uploads/short-url/v27pcxSy7qGzGqXiucYgtUNnCtF.pdf.

[8] Qorvo. *Qorvo UWB Technical Document*.
https://forum.qorvo.com/uploads/default/original/1X/6283704dedcef71b33ac38e682e6a17973aaf143.pdf.

[9] Qorvo Forum. *DW3000 Indoor Positioning Timeout Error Issues*.
https://forum.qorvo.com/t/dw3000-indoor-positioning-timeout-error-issues/17134.

[10] Qorvo Forum. *DW3000 Getting Continuous SFD Timeout RXSTO Events on RX Enable*.
https://forum.qorvo.com/t/dw3000-getting-continuous-sfd-timeout-rxsto-events-on-rx-enable/12272.

[11] Qorvo Forum. *DW3000 Responder Freezes in dwt_starttx during DS-TWR*.
https://forum.qorvo.com/t/dw3000-responder-freezes-in-dwt-starttx-during-ds-twr/22757.

[12] Qorvo Forum. *Receive Frame Wait Timeout Event RXRFTO Not Set*.
https://forum.qorvo.com/t/receive-frame-wait-timeout-event-rxrfto-not-set/8891.

[13] Qorvo Forum. *DW3000 Settings*.
https://forum.qorvo.com/t/dw3000-settings/20801.

[14] Qorvo Forum. *0 cm Distances with TWR When Multiple Anchors Are in the Same Room*.
https://forum.qorvo.com/t/0cm-distances-with-twr-when-multiple-anchors-are-in-the-same-room/14215.

[15] Qorvo Forum. *Custom RTLS with DWM1000 and ESP32-S3: Seeking PANS Firmware or Alternative Solutions*.
https://forum.qorvo.com/t/custom-rtls-with-dwm1000-and-esp32-s3-seeking-pans-firmware-or-alternative-solutions/20322.

[16] Qorvo Forum. *Tags TWR Strategy*.
https://forum.qorvo.com/t/tags-twr-strategy/5512.

[17] Qorvo Forum. *TX after Double Buffer RX Fails*.
https://forum.qorvo.com/t/tx-after-double-buffer-rx-fails/3065.

[18] Qorvo Forum. *DW3000 Multi-Anchor 1-Tag*.
https://forum.qorvo.com/t/dw3000-multi-anchor-1-tag/24557.

[19] Qorvo Forum. *DWM3000 Collision Avoidance on Low-Power Anchor System*.
https://forum.qorvo.com/t/dwm3000-collison-avoidance-on-low-power-anchor-system/22935.

[20] Qorvo Forum. *Seeking Guidance on ESP32 DW3000 Antenna Delay Calibration Process*.
https://forum.qorvo.com/t/seeking-guidance-on-esp32-dw3000-antenna-delay-calibration-process-autocalibration/15390.

[21] Makerfabs. *ESP32 UWB DW3000*.
https://wiki.makerfabs.com/ESP32_DW3000_UWB.html.

[22] Makerfabs. *Makerfabs ESP32 UWB DW3000*.
https://github.com/Makerfabs/Makerfabs-ESP32-UWB-DW3000.

[23] How2Electronics. *Ranging & Localization with ESP32 UWB DW3000 Module*.
https://how2electronics.com/ranging-localization-with-esp32-uwb-dw3000-module/.

[24] How2Electronics. *ESP32 DW3000 UWB Module Achieving 500m Range*.
https://how2electronics.com/esp32-dw3000-uwb-module-achieving-500m-range/.

[25] Espressif Systems. *Arduino Core for ESP32 Documentation*.
https://docs.espressif.com/projects/arduino-esp32/en/latest/.

[26] ESP32Async. *ESPAsyncWebServer*.
https://components.espressif.com/components/esp32async/espasyncwebserver.

[27] ESP32Async. *AsyncTCP*.
https://components.espressif.com/components/esp32async/asynctcp/.

[28] Open Robotics. *ROS 2 Documentation*.
https://docs.ros.org/en/jazzy/.

[29] Open Robotics. *Gazebo Sim Documentation*.
https://gazebosim.org/libs/sim/.

[30] Python Software Foundation. *Python Documentation*.
https://docs.python.org/.

[31] Python Software Foundation. *Requests: HTTP for Humans*.
https://requests.readthedocs.io/.

[32] National Library of Medicine. *Ultra-Wideband Technical Paper*.
https://pmc.ncbi.nlm.nih.gov/articles/PMC10575093/.

[33] Brunner and Boano. *UWB and Wi-Fi Technical Paper*.
https://www.carloalbertoboano.com/documents/brunner22uwbwifi.pdf.

[34] Stocker and Boano. *MSWiM Technical Paper*.
https://www.carloalbertoboano.com/documents/stocker25mswim.pdf.

[35] MDPI Sensors. *Sensors Article on Ultra-Wideband*.
https://www.mdpi.com/1424-8220/22/4/1643.

[36] Graz University of Technology. *Ultra-Wideband Technical Paper*.
https://tugraz.elsevierpure.com/ws/portalfiles/portal/92476252/3666025.3699372.pdf.

[37] Corbalan. *CRNG: Technical Paper*.
https://pablocorbalan.com/files/papers/crng-ewsn18.pdf.

[38] Wikimedia Commons. *Symmetric Double-Sided Two-Way Ranging Diagram*.
https://upload.wikimedia.org/wikipedia/commons/c/cf/Sds-twr.png.

[39] ResearchGate. *Impulse Response of UWB Channel with Non-Overlapping Clusters*.
https://www.researchgate.net/publication/26600404/figure/fig1/AS%3A394306230734850%401471021339622/mpulse-Response-of-UWB-Channel-with-nonoverlapping-clusters.png.

[40] MDPI. *Information Journal Diagram*.
https://pub.mdpi-res.com/information/information-16-01033/article_deploy/html/images/information-16-01033-ag-550.jpg.