
Institute for Computer Science XVII
Robotics

*Master's thesis*

# An EKF-SLAM based Ultra Wideband Localization Approach for Unknown Anchor Distributions

Martin Cosmas Hesse

December 2024

First reviewer: Prof. Dr. Andreas Nüchter
Second reviewer: Prof. Dr. Dorit Borrmann

## Zusammenfassung

Der immer größer werdende Einfluss, den mobile Roboter nicht nur auf die wissenschaftliche Forschung, sondern auch auf das tägliche Leben haben, wird durch ihre präzise Lokalisierung unterstützt. Da in vielen Szenarien, wie zum Beispiel der Planetenerkundung, ein (teil)-autonomes Arbeiten der Roboter erforderlich ist, muss das Lokalisierungssystem dies unterstützen, das bedeutet der Roboter muss sein eigenes Lokalisierungssystem mitbringen. In Situationen, in denen Global Navigation Satellite Systems (GNSS) nicht verfügbar ist, ist die Ultra Wideband (UWB) Technologie ein gängiger Ersatz. Diese setzt in der Regel voraus, dass die Positionen von verteilten Ankern im Voraus bekannt sind. In dieser Arbeit stellen wir ein System vor, das in der Lage ist, die Anker selbst an zufälligen Positionen zu verteilen, ohne die Position kennen zu müssen. Das bedeutet, dass das System vollständig autark eingerichtet wird. In unserem System trägt ein Roboter drei UWB-Tags bei sich, die Two-Way-Ranging (TWR) Messungen mit allen anderen Knoten im System durchführen. Die Abstände der drei Tags zu einem beliebigen Anker werden verwendet, um die relative Position dieses Ankers zum Roboter zu bestimmen. Dies geschieht unabhängig für alle verteilten Anker. Die sich daraus ergebenden relativen Positionen werden als Landmarken in einem Extended Kalman Filter (EKF)-Simultaneous Localization and Mapping (SLAM) Algorithmus interpretiert. Dieser kombiniert die UWB-Messungen mit der Rad-Odometrie, um die Pose des Roboters und die Positionen der Anker zu bestimmen. Unsere Experimente zeigen vielversprechende Ergebnisse in einer Anordnung mit vier Ankern, die deutlich besser sind als die Rad-Odometrie allein. Das System ist auch in der Lage, während eines Ausfalls der UWB-Anker zu funktionieren. Nach einem solchen Ausfall wird in mehreren Experimenten die Position des Roboters erheblich korrigiert, wenn auch nicht auf denselben Standard wie vor dem Ausfall.

# Abstract

The ever-growing influence, that mobile robots have on not just scientific research but also day-to-day life, is supported by their precise localization. As (semi)-autonomous operations of the robots is required in many scenarios, such as planetary exploration, the localization system has to support this, which means the robot has to bring its own localization system along with it. In situations, where Global Navigation Satellite Systems (GNSS) is unavailable, Ultra Wideband (UWB) technology is a common replacement. This typically relies on the positions of distributed anchors to be known beforehand. In this thesis we introduce a system, that is capable of distributing the anchors itself at random positions, without knowing them. This means the system is set up entirely remotely. In our system a robot carries three UWB tags with it, which preform Two-Way-Ranging (TWR) measurements with all other nodes in the system. The distances from the three tags to any anchor are used to determine the relative position of that anchor to the robot. This is done independently for all distributed anchors. The resulting relative positions are interpreted as landmarks in an Extended Kalman Filter (EKF)-Simultaneous Localization and Mapping (SLAM) algorithm. This combines the UWB measurements with wheel odometry to keep track of the pose of the robot and the position of the anchors. Our experiments show promising results in a setup with four anchors, performing considerably better than the wheel odometry on its own. The system is also capable of operating through an outage of the UWB anchors. After such an outage in multiple experiments the pose of the robot is considerably corrected, though not to the same standard as before the outage.

# Danksagung

Den Zeitpunkt des Abschlusses meines Studiums, der durch diese Master Arbeit gekennzeichnet ist, möchte ich zum Anlaß nehmen, den Personen zu danken, die sowohl durch die Unterstützung dieser Arbeit, aber auch außerhalb davon meine Studienzeit positiv geprägt haben. Allen Voraus gilt mein Dank natürlich mienen Betreuern dieser Arbeit Prof. Andreas Nüchter und Prof. Dorit Borrmann, die mich bei der Arbeit stets konstruktiv unterstützt haben und immer wieder neue Ideen und Konzepte angeregt haben. Beide haben mir auch schon vor der Arbeit im Rahmen des Studiums immer wieder Möglichkeiten geschaffen, die mir Freude bereitet haben und mein Studium immens bereichert haben. Auch den anderen Mitarbeitern des Robotik Lehrstuhls möchte ich danken, die Fragen beantwortet haben und mir immer wieder bei der Arbeit geholfen haben. Zuletzt geht mein Dank natürlich auch an meine Familie und Freunde, die mich bei der Arbeit motiviert haben. Insbesondere Caro, die meine Studienzeit nicht nur an der Uni, sondern auch abseits davon zu etwas ganz Besonderem gemacht hat.

# Contents

# Acronyms

**ADS-TWR** Asymmetric Double Sided Two-Way-Ranging.

**AoA** Angle of Arrival.

**API** Application Programming Interface.

**BPM** Burst Pulse Modulation.

**BPSK** Binary Phase-Shift Keying.

**DoF** Degrees of Freedom.

**DS** Direct Sequence.

**DS-TWR** Double Sided Two-Way-Ranging.

**EKF** Extended Kalman Filter.

**FH** Frequency Hopping.

**GNSS** Global Navigation Satellite Systems.

**IC** Integrated Circuit.

**ICP** Iterative Closest Point.

**IMU** Inertial Measuring Unit.

**IR-UWB** Impulse Radio Ultra Wideband.

**LiDAR** Light Detection and Ranging.

**LOS** Line of Sight.

**OOK** On-Off Keying.

**PAM** Pulse Amplitude Modulation.

**PCA** Principal Component Analysis.

**PCB** Printed Circuit Board.

**PPM** Pulse Position Modulation.

**PRF** Pulse Repetition Frequency.

**PSK** Pulse-Shift Keying.

**PSO** Particle Swarm Optimization.

**RFID** Radio Frequency Identification.

**RMSE** Root Mean Square Error.

**RODOS** Real time Onboard Dependable Operating System.

**ROS** Robot Operating System.

**RSSI** Received Signal Strength Indicator.

**SDS-TWR** Symmetric Double Sided Two-Way-Ranging.

**SKITH** Skip the Harness.

**SLAM** Simultaneous Localization and Mapping.

**SS-TWR** Single Sided Two-Way-Ranging.

**TDoA** Time Difference of Arrival.

**TH** Time Hopping.

**ToA** Time of Arrival.

**TOF** Time of Flight.

**TWR** Two-Way-Ranging.

**UAV** Unmanned Aerial Vehicle.

**USB** Universal Serial Bus.

**UWB** Ultra Wideband.

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Mobile robots play an ever-growing role not just in scientific research, but also more mundane aspects of life. They have the ability to explore, inspect, map and transverse areas, that are too dangerous, difficult or remote for humans to access. For many of these tasks (semi)-autonomous operations are required, which rely on localization of the robot. On many places on Earth this can be accomplished using Global Navigation Satellite Systems (GNSS), but there are areas where this is infeasible, including in planetary exploration outside of Earth. For this other forms of localization are necessary, which ideally do not have to be set up in advance, but can rather be put in place by the robot itself during its operation. Simultaneous Localization and Mapping (SLAM) approaches are one such option, which rely on sensors like laser scanners, sonar or Light Detection and Ranging (LiDAR), which output the bearing and range to certain features, whose change is used to determine the pose of the robot. Another option are Ultra Wideband (UWB) sensors, which send messages in between each other and record timestamps of the transmit- and receive-times. Using these the Time of Flight (TOF) is calculated, using a process called Two-Way-Ranging (TWR). The TOF multiplied with the speed of light creates a network of known distances in between each of the UWB nodes. If the position of the fixed sensors (anchors) is known, the position of a moving sensor (tag) is calculated, similar to GNSS.

We have designed a mobile system that distributes multiple UWB sensors using small $CO_2$ powered 3D-printed rockets, to create a network of these sensors, which are used for the localization of the robot. [1] This enables us to set up this localization system autonomously wherever the robot goes, although the position of the anchors after the rocket launch is unknown. We tested the distribution of the sensors and a first iteration of the positioning system at the ESA-ESRIC Space Resources Challenge in 2021 [2] and more recently in 2024 as part of the AMADEE-24 Mars analog mission of the ÖWF in Armenia [3]. We use the DecaWave DWM1000 chip [4] as the UWB sensor. These are integrated onto a custom Printed Circuit Board (PCB), which uses a microprocessor to run the necessary embedded code for the TWR process. This code is based on the operating system Real time Onboard Dependable Operating System (RODOS) [5], which provides an easy connection to Robot Operating System (ROS) [6] running on a centralized PC, via a RODOS-ROS-Bridge. The positional data from the UWB system is then fed into a Kalman Filter [7], where it is combined with odometry data both directly from the wheels of the robot, and visual from an Intel T265 stereo camera [8]. The

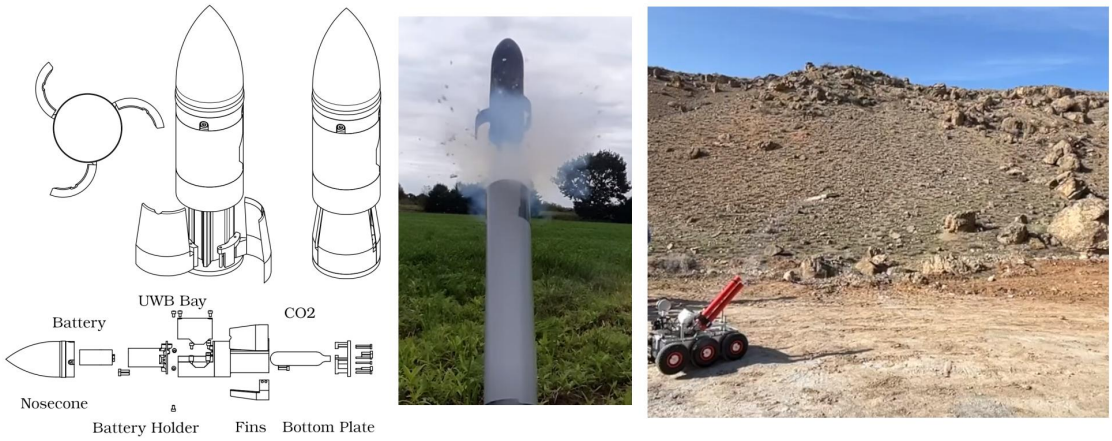



**Figure 1.1:** Design of the rockets, including the $CO_2$-Cartridge propulsion system (left), rocket launch (middle), System tests during AMADEE-24 Mars analog mission (right). [1]

Kalman Filter provides us with an estimation for the location and heading of the robot.

A significant problem with this approach compared to a more standard position determination scheme with UWB sensors, that this thesis aims to solve, is the unknown position of the anchors, which are usually required to be known a priori. Also, only the position and not the heading of the robot, that is equipped with the tag, is determined. This thesis presents a novel evaluation system, that relies on three tags placed at known position on the robot. With this additional range information we determine the position of the anchors using trilateration and using the positional information in a SLAM based approach estimates not only the position, but also the heading of the robot. This enables a fully autonomous system for both the distribution and usage of the UWB sensors for localization, which is carried by the robot itself.

## 1.1 Thesis Outline

This thesis is structured in multiple chapters, that piecewise introduce and explain technology and processes, which culminates in their combination in the full localization system. Chapter 2 Related Work introduces the concept of UWB technology and SLAM algorithms, how they work and what research has been done in these fields. The Chapter 3 Theoretical Background picks this up and further details the general functional principle of the technologies used in the localization system, mainly TWR and Kalman Filters. How these are implemented in our system and the robot used is described in Chapter 4 Approach. The localization system is then evaluated based on the experiments in Chapter 5 Experiments and Evaluation. This includes experiments, that test the capabilities of the UWB nodes (Chapter 5.1 - Chapter 5.3), individual parts of the localization system (Chapter 5.4) and finally the full system (Chapter 5.5). Chapter 6 Conclusion summarizes the work we did in this thesis and its results. It also gives an outlook into the future, that highlights work that remains to be done and ways to expand and improve the localization system.

# Chapter 2

# Related Work

In our work we mainly use two technologies, that are already widely in use. The first is UWB communications and the second is SLAM. We combine both of these technologies to create our UWB based localization system. In this chapter we first introduce the UWB technology and present some research, that has been done in the field. In particular about the wide bandwidth signal creation and the ranging process. Then we give an overview of the research on SLAM, and its implementation differences based on paradigm and sensor type.

## 2.1 Ultra Wideband Technology

UWB technology is a general classification for wireless communication schemes, that utilize a larger than usual bandwidth. While other traditional communication schemes, like Wi-Fi, Bluetooth or cellular networks try to reduce the bandwidth as much as possible, to create space in the frequency spectrum for as many unique channels as possible, the bandwidth of the transmitted signal with UWB communication schemes is generally at least 500 MHz. Also, the used frequencies for UWB transmission are widely spread out from 3.1 GHz to 10.6 GHz. This is depicted in Figure 2.1. From the graphic we also notice the much lower emitted signal power. UWB signals are usually strictly limited to -41.3 dBm/MHz by the responsible government agencies [11–15]. This very low signal intensity results in a potential UWB signal, that overlaps with another wireless communication signal, acting like regular background noise to it [12, 13]. Therefore, UWB technology is used in many applications, regardless of restrictions, due to other communication systems. The obvious disadvantage to the technology is the comparative short range of the signals, usually at most a couple of hundred meters [4]. But there are many advantages, that come with the unique design. Due to the low signal power, very little energy is needed for the transmission of UWB messages. This makes the system highly energy efficient and allows for flexible data rates, depending on the requirements of the exact use case [11]. [12] also highlights the simplified transceiver architecture, which also helps to keep the cost of UWB Integrated Circuit (IC)s low. Finally, [13, 16] note, that UWB signals have a greater ability to penetrate into obstacles avoid them entirely through mulitpathing.

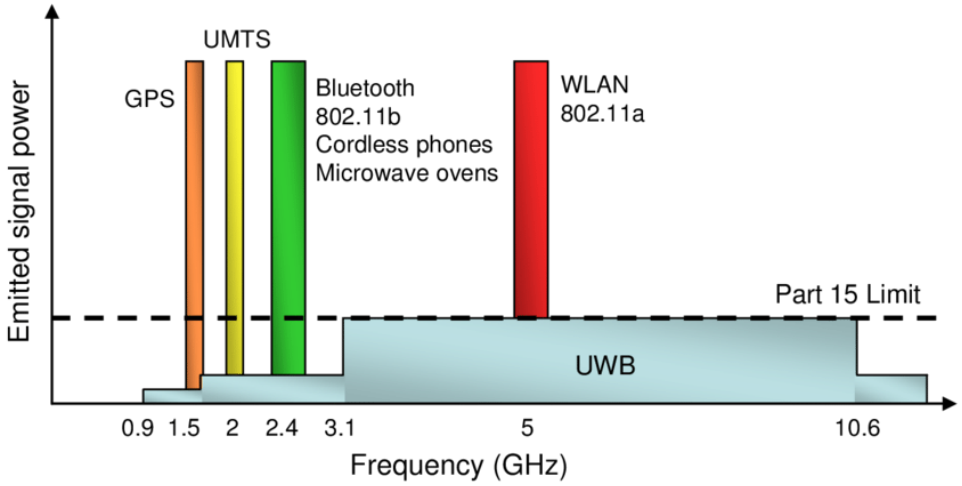



**Figure 2.1:** An overview of the frequencies used by UWB communication schemes, compared to other applications. [9]

### 2.1.1 Signal Creation

To create the large bandwidth of the signal, the creation process of the signal is unique. According to [16] there are three main options to create an UWB signal, that accomplish the large bandwidth. The options are Frequency Hopping (FH) where a signal is transmitted on many frequencies simultaneously, Time Hopping (TH) where very short pulses are used and Direct Sequence (DS) where a signal is spread over a large bandwidth.

**Frequency Hopping** As the name suggests this technique occupies the large bandwidth by changing the frequency, at which the signal is transmitted. During the transmission the frequency is changed with in interval $T_{hop}$, thereby iterating through the entire bandwidth [16]. The hop time $T_{hop}$ can be smaller than the symbol rate ('fast hopping'), larger than the symbol rate ('slow hopping'), or equal [16]. In a variation the transmission frequency is continuously changed covering the full bandwidth. This is called chirp signaling [16, 17].

**Time Hopping** With time hopping instead of hopping through frequencies, the signal hops through time, meaning only transmitting in short bursts [16]. [18] calls this Time Modulated UWB, whereas [11] calls it Impulse Radio Ultra Wideband (IR-UWB), which is nowadays the more common term. For this sub nanosecond radio pulses are created, which usually are formed of a higher order Gaussian peak, which due to its very short width in the time domain, occupies a large width in the frequency domain [11, 16, 18]. Such a pulse is shown in Figure 2.2. Due to the short pulses this signal creation method has a duty cycle of less than 1%, which reduces its power consumption compared to the other methods [11]. The data is modulated on to the signal of short pulses using one of many commonly used modulation schemes, like Pulse Amplitude Modulation (PAM), On-Off Keying (OOK), Pulse Position Modulation (PPM), Pulse-Shift Keying (PSK) or Binary Phase-Shift Keying (BPSK) [11, 12]. The signal is then transmitted directly at the chosen frequency, without the use of a carrier wave. [11] also states a higher accuracy of this

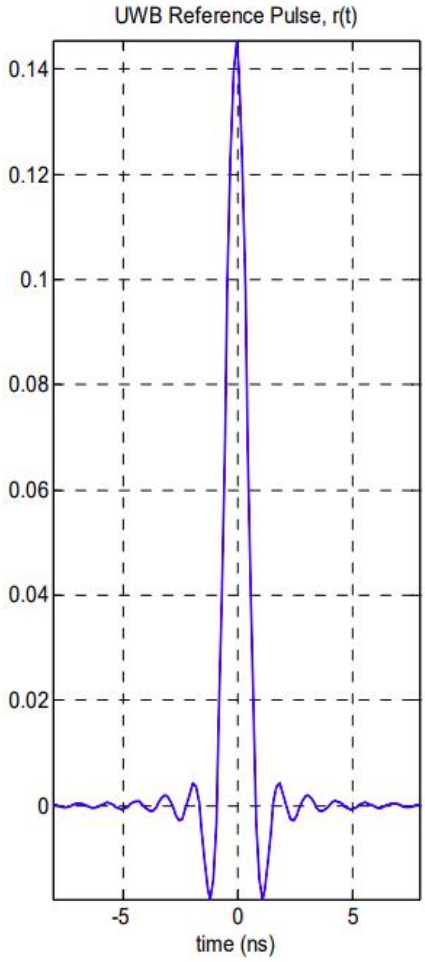



**Figure 2.2:** An IR-UWB reference pulse. [10]

method for range finding applications, like the one we implement in this thesis. According to [13, 16] the short pulses used allow for greater resilience against multipath problems, as the receiver can clearly distinguish between them. A multipath detection and suppression feature is also implemented in the DWM1000 chip we use, that is explained in more detail in Chapter 4.2.1 [19]. The low duty cycle of this implementation has a negative effect on the time synchronization, making this more difficult for use cases that require synchronization.

**Direct Sequence** DS or Frequency-Modulation UWB in [11] use a wideband frequency modulation of a baseband signal, instead of a short pulse. [16] describes the process as follows: First a normal baseband signal is created, which then spread over a large bandwidth. For this a pseudo random sequence is multiplied on to the signal, thus spreading it over the large bandwidth. The receiver of the signal correlates is with the same spreading sequence to retain the original baseband signal and the corresponding data. [11] describes a process similar to a two-step

double frequency modulation scheme. First the baseband subcarrier generation applies BPSK to create a triangular wave at two distinct frequencies. Then using the voltage of a voltage controlled oscillator radio frequency modulation spreads the signal out over a large bandwidth. As the voltage peak of the voltage controlled oscillator is not very high, this suits low power applications such as UWB communications. The continuous modulation of the signal leads to a higher duty cycle, compared to IR-UWB. This reduces the energy efficiency of this method, but instead it offers higher sensitivity. Also, the transmission range and data rate with DS are lower than with IR-UWB. [11]

Overall it is imperative to choose the correct system to create the UWB signal. Nowadays most UWB implementations either use IR-UWB or DS depending on which suits the requirements of their main use case better [11]. [16] suggests, that a hybrid solution combining the multiple approaches is feasible.

### 2.1.2 Use Cases

Due to their many advantages, especially the high energy efficiency for short range communications, UWB technology has a wide set of use cases [11, 13, 16]. We categorize the these in three main functions, although each of these contain a large amount of applications:

1. **High data rate:** Using the full capabilities of the UWB technology allows for data rates of hundreds of Mbps. This is considerably higher than Radio Frequency Identification (RFID) communication or Bluetooth technology, which transmit at most hundreds of Kbps or single digit Mbps respectively, and is only surpassed by Wi-Fi technology, which can at most transmit in the Gbps range [20–22]. With this UWB technology is ideally suited for high data rate, low range communications [11, 12, 16]. One example of a system, that aims to replace heavy and convoluted wiring harnesses with UWB is the Skip the Harness (SKITH) project [23]. In fact the custom designed PCBs for that project are the ones, that we use in this thesis.

2. **Low data rate:** There are many low data rate applications, that require long lifespan and easy maintenance, which the low energy usage if UWB technology enables. These applications include equipment tracking, inventory control and others, like in the Apple AirTags. [12]

3. **Ranging:** Besides for the communication aspect UWB technology is also used to preform ranging operations. With this not the data transmitted by the signal, but rather the signal itself is used determine the distance between two UWB nodes. For outdoor positioning GNSS has long been the obvious and widespread solution, but it faces significant problems in areas, where there is no GNSS signal, like indoors, caves or planetary exploration. UWB technology is ideally suited to fill this gap, by preforming positioning with some fixed distributed anchors at known locations and a moving tag. [11, 13, 16]

As we use the UWB technology to preform localization of a robot using ranging in between the nodes, we introduce the common options for the distance measuring. [11] states, that there are five general options two preform ranging: Received Signal Strength Indicator (RSSI), Time

of Arrival (ToA), Time Difference of Arrival (TDoA), Angle of Arrival (AoA) and TWR. Three of which are typically used with UWB for the positioning of a tag:

**Two-Way-Ranging** TWR uses an exchange of multiple messages between two nodes to determine their distance. It saves timestamps of the tx- and rx-times, from which the TOF is calculated, and from that the distance. In the simplest form this method requires clock synchronization between the nodes, but this is circumvented by using more complex evaluation methods. As this is the method used in our implementation a more detailed overview of the TWR process and its evaluation is given in Chapter 3.2. Due to the multiple messages this method has higher energy consumption, than other methods, but it has higher precision. [10, 11]

**Time Difference of Arrival** Instead of an exchange of multiple messages TDoA only uses a single message. It is transmitted from the tag to the anchors. At the anchors the time difference of the arrival of the message is noted, from which the relative distances of the anchors the tag is calculated. Using this information the position of tag is determined. To measure the time difference precisely the clocks of the anchors have to be synchronized, which complicates the process. As an advantage the fewer messages sent reduce the energy usage of this method, as well as lowering the latency. [11]

**Angle of Arrival** Whereas all other ranging methods measure either absolute or relative distances, from which the overall system geometry is recreated, AoA measures the angle of an UWB message. It utilizes a fixed pair of antennas with a small known distance in between, which can be substituted by a single rotating antenna. To calculate the angle either the distances from the tag to both of the antennas is measured, or the difference of the respective arrival time, if the antennas are connected to the same clock for synchronization. Using this and the known distance between the antennas the angle is determined. Due to the generally small distance between the antenna, this method is greatly affected by small deviations in the distance measurement. This leads to a low accuracy of the positioning method, if used alone. Therefore, this is typically used as a complement to another ranging technique or multiple antenna pair stations are set up. [11]

## 2.2 Simultaneous Localization and Mapping

SLAM is a technique widely used for the localization of mobile robots. At the same time it enables the robot to create a map of its environment, which gives it the ability to explore autonomously. It also provides a positioning system, that is independent of GNSS availability, and is therefore widely used in robotics, autonomous cars, mapping applications and more. The first mention of SLAM was in [25], but there has been some research, that used the same systematic ideas for autonomously guided vehicles, in the years prior, e.g. [26]. In Figure 2.3 [24] gives an overview of the general process of any SLAM algorithm. The core of any SLAM algorithm consists of two parts: First a front-end, that is responsible for feature extraction and tracking in the supplied data. This data comes from a selection of a set of visual and LiDAR based sensors on the robot. And second the back-end, which estimates both the map and pose

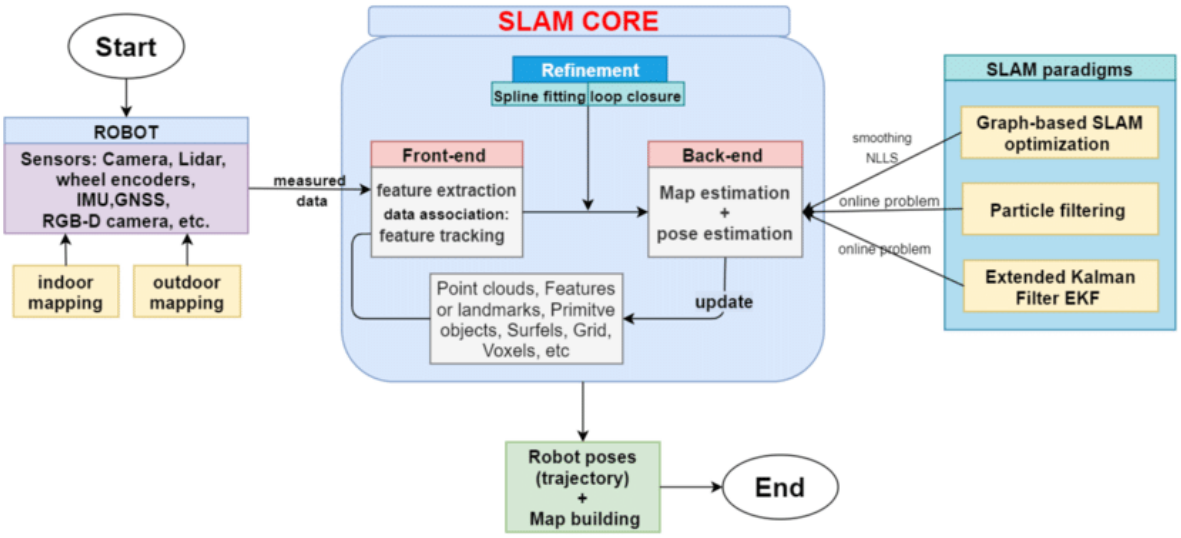



**Figure 2.3:** An overview of the components and techniques in SLAM algorithms. [24]

of the robot. While the design of the front-end is mainly dependent on the type of sensing technique, the back-end is mainly differentiated by the SLAM paradigm it uses. [24, 27–32]

### 2.2.1 SLAM Paradigms

The SLAM paradigm used and its implementation is the heart of any SLAM algorithm. The implementation is categorized in one of two options: Either a filter based approach, in particular an Extended Kalman Filter (EKF), or a graph based SLAM. [24, 29, 30, 32]

**EKF-SLAM** The oldest SLAM algorithms are all based on an EKF. The state of the EKF combines the pose of the robot with the location of the features detected in the incoming data from the sensors on the robot. It uses the probabilistic properties of the sensor readings to set up the filter. Usually an Inertial Measuring Unit (IMU) or wheel odometry is used as the input of the EKF, which creates a state prediction. This is verified by the main visual or LiDAR based sensor input of the SLAM algorithm. One main disadvantage of the EKF-SLAM is its high computational demands. Even though there has been research in to the reduction of the computational complexity and computers have greatly increased computational power, EKF-SLAM is getting less popular in modern systems. This is also due to the fact, that it requires a full motion and observation model of the robot, which is not always given, and wrong landmark association has a large negative effect on it. [24, 25, 30, 32]

**Graph SLAM** A graph based SLAM algorithm was first introduced in [33]. In the graph the vertices represent the pose of the robot and the landmarks, similar to the state in the EKF. The edges connecting the vertices represent the sensor measurements, which describe a transition from one pose to another, both between the robot and the landmarks, but also

in time between previous poses and the most recent one. More precisely the edges encode a probability distribution between the different poses. Obviously there exists no perfect fit for the measurements due to unavoidable noise in the measurement system. The goal of the SLAM algorithm is to find a pose, that maximizes the consistency of the edges and nodes, to the actual measurements. This optimization problem is typically solved using a least squares error minimization procedure. Most modern research on SLAM is in the further optimization of the graph fitting problem. [24, 29, 32]

### 2.2.2 Sensing Technique

The type of sensor used by the robot for the SLAM algorithm has a significant impact on the overall design of the SLAM core, in particular the front-end is effected. For this data input the many applications, that utilize SLAM, choose a wide range of sensors, depending on the exact scenario and environment the robot faces. The sensing technique is divided into two main groups, visual sensors, such as cameras and LiDAR sensors, such as laser scanners. While traditionally many SLAM systems relied on LiDAR sensors, cameras are getting more popular, even in highly complex scenarios, such as self-driving cars. [24, 32]

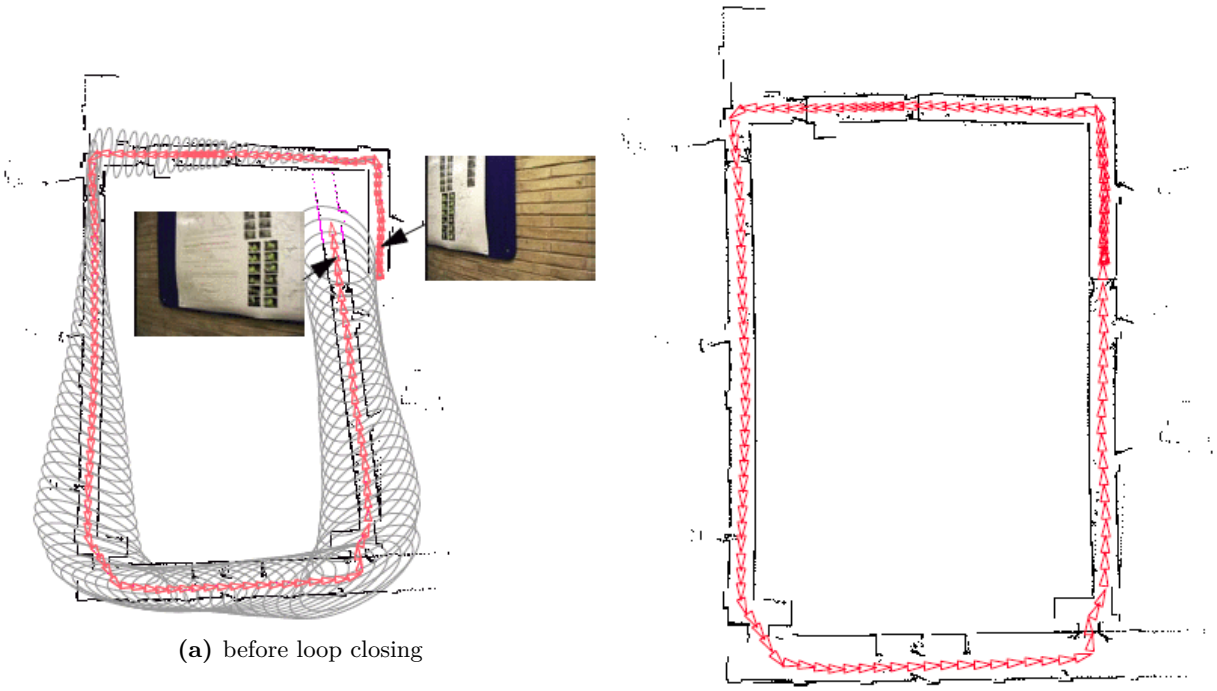
**Visual SLAM** Visual SLAM is based on passive sensors, that capture external light, mainly cameras. This also highlights a large disadvantage, they require external light to function, so in permanently dark environments or during nighttime their capability is greatly reduced. But this system comes closest to human eyes, which is what humans mainly use to observe their surroundings and plan its traversal. With the use of multiple cameras with a certain offset this also allows gathering depth information, just like humans would lose spatial awareness, if one eye is covered up. One great advantage of visual systems is the relative low cost of the sensors, compared to LiDAR. They work well in highly dynamic environments, especially those, that are feature rich. Compared to LiDAR sensors they also have the ability to differentiate between different colors, so they are not limited to the shape of objects. The front-end of the visual SLAM core is responsible for feature extraction out of the image and feature matching to determine the change of the robots pose. Feature extraction is done in one of two ways. Either a feature point method is used, in which striking points in the image are detected and tracked over time. Popular methods include GTFF [34], Harris [35], FAST [36], SIFT [37], SURF [38] and ORB [39]. Or the direct method is used [40]. Instead of prominent features in the image, a grayscale of it is used to track the movement over time. This requires some shaded and non shaded areas in the image to work well, but it also works in feature-poor areas, like blank hallways. The feature extraction and tracking is used to estimate the transition from the current pose to the next pose of the robot. The back-end then combines this estimation with the current pose to update the overall pose estimation. It also combines the images to create a map of the explored area and is responsible for other unique aspects of the SLAM algorithm such as loop closing. [24, 32]

**LiDAR SLAM** The other large group of sensors used for SLAM are LiDAR sensors. Overall this is the more mature sensor type, which has been used with SLAM since its conception. These are active sensors, that send out a laser, or some other form of ray, that is reflected by

the robots surrounding and detected by the sensor. With the delay between sending out the ray and its detection, the distance to the closest object in the particular direction is determined. By rotating the sensor this allows it to create a point cloud of the entire surrounding of the robot. It also enables the system to work in the dark and other advisory conditions for cameras, such as fog, as this does not block the lasers. The overall task of the front- and back-end is the same as with visual SLAM, but the implementation is somewhat different. In the front-end there is no longer a need for feature extraction, consecutive scans are simply aligned to one another, with algorithms such as Iterative Closest Point (ICP) [41]. The relative change in the pose of the robot is then used in the back-end to update the overall pose estimation. Looking at only the change between two consecutive scans, or images in the case of visual SLAM, produces some cumulative error, that manifests itself in the form of drift over time. To counteract the drift LiDAR based SLAM has the opportunity to repeatedly match two scans that are further apart, both time and location wise. This is done by the back-end, to reduce the cumulative error, as now there is not only a relative connection to the directly previous pose, but also to poses further back. This enables LiDAR based systems to be more accurate in the long term compared to visual SLAM, especially when the there is no other positioning system, that works in a global frame, such as GNSS. The overall better point cloud matching capability with LiDAR data also increases the quality of the resulting map. This makes it superior to visual SLAM especially in use cases, where the resulting map is a crucial part of the overall system goal. Overall the advantages of LiDAR based SLAM come with considerably higher cost of the required sensors on the robot. [24, 32]

A vital part of any SLAM implementation is loop closing. This is a procedure usually done by the back-end of the SLAM core, which is necessary whenever an area is left and later reentered, creating a loop in the robot trajectory. If done properly, it has the potential to greatly increase the long term accuracy of both the pose estimation and the created map. Once the robot reenters a previously visited area, the content in the images or scans depending on the sensor type, obviously repeat. But due to drift, the pose estimation of the robot no longer aligns with the reality, which not only leads to a continuation of the drift, but also to a doubling of the created map. This phenomenon is shown in Figure 2.4a, where the map is doubled to the left of the map created at the beginning of the test. To preform the loop closing frames, either visual or LiDAR based, are matched to previously taken keyframes to detect the overall system drift. With this the pose estimation of the robot is corrected along the entire past trajectory accounting for the drift of the system. Also, the two parts of the map are connected to create a continuous map of the robots surroundings. The successful result of loop closing is depicted in Figure 2.4b. [24, 32, 42]

In our work we build on the research done in the field of ranging with UWB technology, to localize a robotic system. To preform the ranging we utilize TWR, as this does not require any clock synchronization and has higher accuracy. For the actual localization we implement an EKF-SLAM algorithm, that uniquely does not use either visual or LiDAR sensors as an input, but rather the UWB anchors as landmarks. As the number of features is very limited in this approach, simply the number of anchors, the state of the EKF is not particularly large, thus also keeping computational requirements low. Also, we do have a full motion model of the robot



**(a)** before loop closing

**(b)** after loop closing

**Figure 2.4:** Snapshots of a SLAM algorithm without and with loop closing. [42]

and the landmarks and there is no problem with landmark association, as the UWB anchors transmit unique IDs, eliminating the need of feature extraction. With this the main problems with an EKF based SLAM system are not relevant to our implementation. We do also use a 2D visual SLAM approach for the creation of the ground truth.

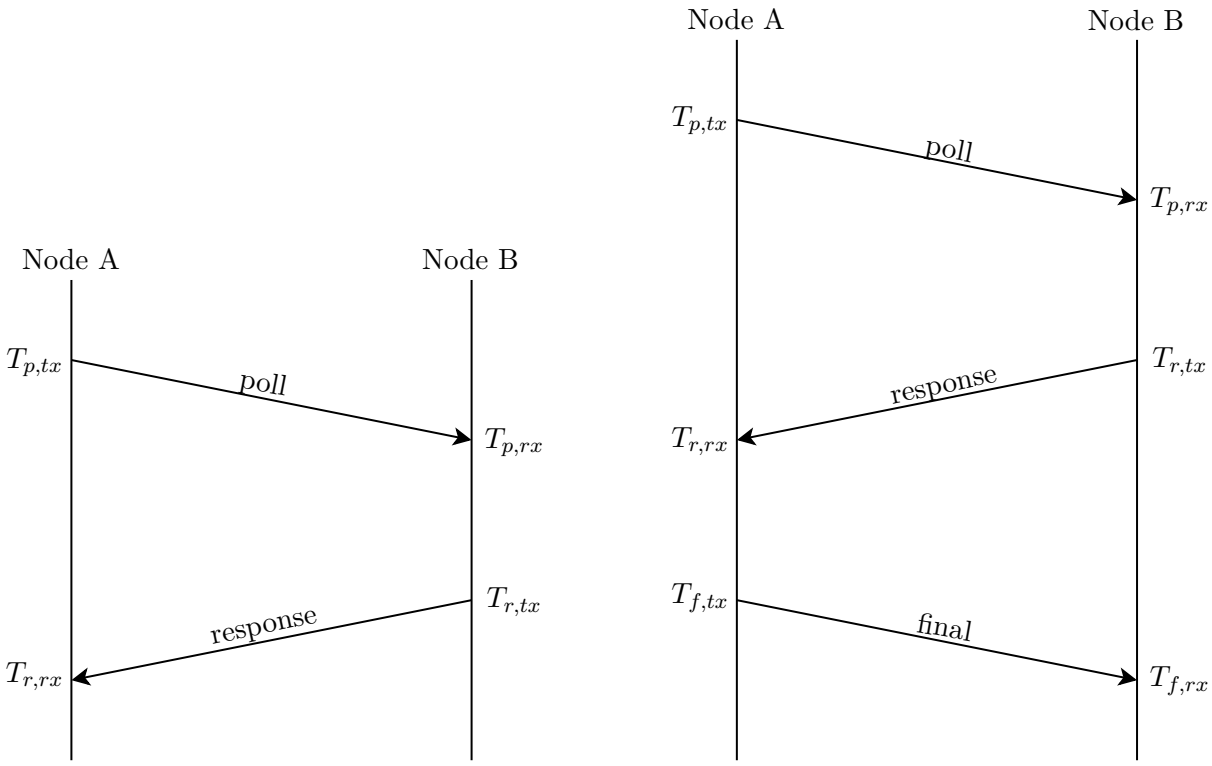
# Chapter 3

# Theoretical Background

In this chapter we describe the relevant processes necessary for our Approach. The exact setup and usage of these processes in our Experiments is explained in detail in Chapter 4, whereas this gives an explanation of the general workings of them. This includes the usage of the specific type of UWB messages, created by the chip used in our implementation, and how we use the messages in a TWR process to calculate the distance between two such nodes. The distance measurements alongside the wheel odometry then get input into a EKF, whose workings are also detailed here. Besides this, we also introduce the ICP algorithm, which is used to find a transformation between multiple laser scans, which will provide us with a ground truth to compare the results of the UWB localization system to.

## 3.1 UWB Messages

UWB messages are a low energy, low rate communication scheme, that occupy the frequency bands from 3.1 GHz to 10.6 GHz, so a much larger part of the spectrum, than any other communication schemes. A comparison of commonly used frequency bands and the UWB frequency bands is depicted in Figure 2.1. Within this band the messages only emit a very low amount of energy, usually capped at -41.3 dBm/MHz by the responsible government agencies [14, 15]. The current IEEE Standard [10] governing Low-Rate Wireless Networks such as UWB networks defines multiple channels within these frequency bands. Each of these channels differentiate each other based on their respective center frequency and their bandwidth. All channels use at least a bandwidth of 500 MHz, in fact most of them use this bandwidth. As the name UWB suggests this is a much larger bandwidth than used by other communication schemes. This allows the individual pulses to transmit a large enough amount of energy to be detected, while still emitting only a very low amount of energy per Mhz, resulting in minimal disruptions. The corresponding very short pulses have a length of up to 2 ns, such a reference pulse is shown in Figure 2.2. Other than more standard communication schemes UWB pulse are not modulated onto a carrier frequency, but the very short pulse itself corresponds to the center frequency of the used channel. This means other modulation schemes are used, typically Burst Pulse Modulation (BPM) or BPSK. Each UWB message is made up of multiple of these pulses, organized in three parts. First is the Synchronization Header, responsible for the synchronization of the receiver.




**(a)** SS-TWR **(b)** DS-TWR

**Figure 3.1:** Overview of the messages sent and their respective timestamps during one execution of the varying TWR Schemes.

This consists of a varying amount of non modulated symbols, that are either defined by either a positive, a negative or no pulse. Second is the Physical Header defining the length and data rate of the transmission, while also including bits for forward error correction. Third is payload field, which contains the actual data to be sent. In all further discussions of the UWB messages we only discuss the content of the payload field, as the headers are automatically created by the IC that implements the standard [10].

## 3.2 Two-Way-Ranging

TWR is one option to use the aforementioned UWB messages to determine the distance in between two UWB nodes. It uses the naturally occurring delay (or TOF) of any wirelessly sent message, which is recorded using multiple timestamps. With the speed of light $c_0$ the distance $d$ follows as:

$$d = T_f c_0 \tag{3.1}$$

To get the TOF $T_f$ using the UWB messages there are multiple options.

### 3.2.1 SS-TWR

The simplest is the Single Sided Two-Way-Ranging (SS-TWR) scheme, which only needs two messages to be sent. An overview of this is shown in Figure 3.1a. The initiating node (in the Figure Node A) sends out a poll message and records the transmit-timestamp $T_{p,tx}$ of the exact time, when the message is sent. Once the other node (Node B) receives the poll message it records a receive-timestamp $T_{p,rx}$ and sends a response message back. As the processing time of the node is in the order of milliseconds, whereas the TOF is only a few nanoseconds, the node also has to record an exact transmit-timestamp $T_{r,tx}$ of the response message. The first node sets the final timestamp $T_{r,rx}$ as soon as the response-message is received. As the two nodes run fully independently of one another and have no global time setter, the actual value of the timestamps of each node are completely arbitrary. Therefore, a direct comparison between two timestamps from different nodes will not yield a useful result, meaning only differences between timestamps from the same node offer actual information about the elapsed time. We call the time it takes for a node to respond to a received message reply time $D_b = T_{r,tx} - T_{p,rx}$ and the time it takes to receive a response after a message is sent out round time $R_a = T_{r,rx} - T_{p,tx}$. Using these the TOF is easily calculated by:

$$T_f = \frac{1}{2}\left(R_a - D_b\right) \tag{3.2}$$

A large source of error in the calculation of the TOF is the assumption, that the clocks of the nodes run at the exact same frequency. But this is not the case, as the clocks can drift with up to $\pm 20\,\text{ppm}$ as per [43]. As shown in [43] the error of the TOF is calculated as

$$\Delta T_f = T_f e_a + \frac{D_b}{2}\left(e_a - e_b\right) \approx \frac{D_b}{2}\left(e_a - e_b\right), \tag{3.3}$$

with $e_a$ and $e_b$ representing the clock drift of the nodes A and B respectively. The reply time of a few milliseconds is the dominant factor, and corresponds to an error of tens of nanoseconds, representing a distance error of multiple meters. Therefore, the SS-TWR approach does not provide useful distance measurements. Instead, a different TWR scheme has to be used.

### 3.2.2 DS-TWR

The Double Sided Two-Way-Ranging (DS-TWR) approach expands on the SS-TWR by adding a third UWB message, which is shown in Figure 3.1b. This final message is also timestamped, providing us with $T_{f,tx}$ and $T_{f,rx}$. With this we interpret the response and final messages as a second set of TWR measurements, this time initiated from the other side, hence DS-TWR. From here we once again define the reply time $D_a = T_{f,tx} - T_{r,rx}$ and the round time $R_b = T_{f,rx} - T_{r,tx}$. With this we average the two TWR measurements giving us

$$T_f = \frac{1}{4}\left(R_a - D_b + R_b - D_a\right) \tag{3.4}$$

for the TOF. Just like before the error due to clock drift is calculated as [43]:

$$\Delta T_f = \frac{1}{2} T_f \left(e_a + e_b\right) + \frac{1}{4}\left(e_a - e_b\right)\left(D_b - D_a\right) \approx \frac{1}{4}\left(e_a - e_b\right)\left(D_b - D_a\right) \tag{3.5}$$

As long as the reply times $D_a$ and $D_b$ are equal, the TOF error specified in (3.5) will only be a fraction of $T_f$, which represents an error of less than 1 mm for a distance of 10 m. This evaluation method is called Symmetric Double Sided Two-Way-Ranging (SDS-TWR), as the reply times have to be equal. [44] introduces a new evaluation method Asymmetric Double Sided Two-Way-Ranging (ADS-TWR), which reduces the error even if the reply times are not equal:

$$T_f = \frac{R_a R_b - D_a D_b}{R_a + R_b + D_a + D_b} \tag{3.6}$$

The corresponding error is:

$$\Delta T_f = \frac{T_f}{2} (e_a + e_b) \tag{3.7}$$

From (3.7) it is obvious, that the error due to clock drift is no longer an issue, even if the reply times are asymmetrical, because the error will be at most a small fraction of the TOF. Therefore, we have an evaluation method in (3.6), that is robust against clock drift, which forms the basis for our UWB localization system, by providing accurate distances between the nodes.

### 3.2.3 Antenna Delay

Another error source besides clock drift is antenna delay. For (3.6) to produce accurate data, the timestamps have to represent the transmit- and receive-times exactly. Even though UWB ICs are designed to timestamp incoming and outgoing messages immediately, they have to calibrate for the tx- and rx-antenna delay. For this the tx-antenna delay has to be added to all tx-timestamps and the rx-antenna delay subtracted from all rx-timestamps. As both the reply and round times are each defined by the difference between one tx-timestamp and one rx-timestamp, we do not have to differentiate between the individual tx- and rx-antenna delays, so both are set to the equal antenna delay $\epsilon$. This is added or subtracted from the timestamps respectively, which leads to the following changes to the reply and round times:

$$\forall i \in \{a, b\} :$$

$$R_i^* = R_i - 2\epsilon \tag{3.8}$$

$$D_i^* = D_i + 2\epsilon \tag{3.9}$$

This will change (3.6) to be:

$$\begin{aligned} T_f^* &= \frac{R_a^* R_b^* - D_a^* D_b^*}{R_a^* + R_b^* + D_a^* + D_b^*} = \frac{R_a R_b - 2\epsilon R_a - 2\epsilon R_b + 4\epsilon^2 - D_a D_b - 2\epsilon D_a - 2\epsilon D_b - 4\epsilon^2}{R_a - 2\epsilon + R_b - 2\epsilon + D_a + 2\epsilon + D_b + 2\epsilon} \\ &= \frac{R_a R_b - D_a D_b - 2\epsilon (R_a + R_b + D_a + D_b)}{R_a + R_b + D_a + D_b} = T_f - 2\epsilon \end{aligned} \tag{3.10}$$

From (3.10) we notice that, an increase of the antenna delay results in a decrease of the TOF by twice the increase, similarly the measured distance (3.1) is decreased by the respective amount. To calibrate the system we must determine the antenna delay $\epsilon$ and adapt the timestamps accordingly in our implementation of the TWR process. There are multiple calibration options to determine the antenna delay, some of which, that we have tested, are detailed in Chapter 5.2.

So far we have ignored the fact, that the timestamps are taken on the individual UWB nodes and have to be shared with one node, which actually preforms the calculation based on those timestamps. There are many ways how the timestamp information may be shared with each other, or potentially a non-involved node. In Chapter 4.2.2 we show how in our implementation we share all timestamps with one of the nodes, without requiring any additional messages. So for the functional principle of the TWR process the exchange of timestamps has no effect, it only effects the actual implementation.

A system implementing the ADS-TWR process and using a decent antenna calibration, has the ability to provide distances measurements between any of the distributed UWB nodes. In our implementation these distance measurements are further processed to provide us with meaningful positional data for our particular setup, which is described in Chapter 4.2.3. But this further processing does not have an influence on the general workings of the processes used, therefore it is not explained in this chapter.

## 3.3 Kalman Filter

The positional information gathered from the UWB nodes and the wheel odometry are then fused in an EKF. The general working of which, is described in the following, whereas the exact setup of the EKF used in our experiments is discussed in Chapter 4.2.4. [7, 45]

The Kalman Filter consists of a state $\boldsymbol{x}$ of dimension $n$ and a corresponding covariance matrix $\boldsymbol{P}$ of dimension $n \times n$. In the simplest form the state at a discrete time step $\boldsymbol{x}_k$ has a linear relation to the state at the previous time step $\boldsymbol{x}_{k-1}$ and the system input $\boldsymbol{u}_k$. Also, the process noise $\boldsymbol{w}_k$ is zero mean additive white Gaussian noise with a variance $\boldsymbol{Q}_k$. Similarly, the observation $\boldsymbol{z}_k$ has only a linear dependence on the state $\boldsymbol{x}_k$ and its noise $\boldsymbol{v}_k$ is also zero mean additive white Gaussian noise with a variance $\boldsymbol{R}_k$. Therefore, the next state and the observation are defined as:

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k + \boldsymbol{w}_k \tag{3.11}$$

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k \tag{3.12}$$

The state transition $\boldsymbol{F}_k$, input model $\boldsymbol{B}_k$ and observation model $\boldsymbol{H}_k$ each represent the linear relationship from the old state, the system input and the current state to the current state and the observation respectively. These relations allow us to set up the full Kalman Filter, which consists first of a prediction step and then of an update step.

Prediction:

predicted state: $$\hat{\boldsymbol{x}_k} = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_k \tag{3.13}$$

predicted covariance: $$\hat{\boldsymbol{P}}_k = \boldsymbol{F}_k \boldsymbol{P}_{k-1} \boldsymbol{F}_k^\top + \boldsymbol{Q}_k \tag{3.14}$$

Update:

$$\text{Innovation:} \quad \boldsymbol{y}_k = \boldsymbol{z}_k - \boldsymbol{H}_k \hat{\boldsymbol{x}_k} \tag{3.15}$$

$$\text{Innovation covariance:} \quad \boldsymbol{S}_k = \boldsymbol{H}_k \hat{\boldsymbol{P}}_k \boldsymbol{H}_k^\top + \boldsymbol{R}_k \tag{3.16}$$

$$\text{Kalman Gain:} \quad \boldsymbol{K}_k = \hat{\boldsymbol{P}}_k \boldsymbol{H}_k^\top \boldsymbol{S}_k^{-1} \tag{3.17}$$

$$\text{updated state:} \quad \boldsymbol{x}_k = \hat{\boldsymbol{x}_k} + \boldsymbol{K}_k \boldsymbol{y}_k \tag{3.18}$$

$$\text{updated covariance:} \quad \boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \hat{\boldsymbol{P}}_k \tag{3.19}$$

### 3.3.1 Extended Kalman Filter

The EKF expands on the Kalman Filter, by removing the linearity constraint of the next state and the observation. [45] The functions $\boldsymbol{f}$ and $\boldsymbol{h}$ governing these relations now no longer have to be linear, just differentiable:

$$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k \tag{3.20}$$

$$\boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{v}_k \tag{3.21}$$

The process noise $\boldsymbol{w}_k$ and observation noise $\boldsymbol{v}_k$ remain zero mean additive white Gaussian noise with their respective covariances $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$. The prediction and update step of the Kalman Filter remain similar, although now there are functions $\boldsymbol{f}$ and $\boldsymbol{h}$, instead of the matrices $\boldsymbol{F}$, $\boldsymbol{B}$ and $\boldsymbol{H}$:

Prediction:

$$\text{predicted state:} \quad \hat{\boldsymbol{x}_k} = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{3.22}$$

$$\text{predicted covariance:} \quad \hat{\boldsymbol{P}_k} = \boldsymbol{F}_k \boldsymbol{P}_{k-1} \boldsymbol{F}_k^\top + \boldsymbol{Q}_k \tag{3.23}$$

Update:

$$\text{Innovation:} \quad \boldsymbol{y}_k = \boldsymbol{z}_k - \boldsymbol{h}(\hat{\boldsymbol{x}_k}) \tag{3.24}$$

$$\text{Innovation covariance:} \quad \boldsymbol{S}_k = \boldsymbol{H}_k \hat{\boldsymbol{P}}_k \boldsymbol{H}_k^\top + \boldsymbol{R}_k \tag{3.25}$$

$$\text{Kalman Gain:} \quad \boldsymbol{K}_k = \hat{\boldsymbol{P}}_k \boldsymbol{H}_k^\top \boldsymbol{S}_k^{-1} \tag{3.26}$$

$$\text{updated state:} \quad \boldsymbol{x}_k = \hat{\boldsymbol{x}_k} + \boldsymbol{K}_k \boldsymbol{y}_k \tag{3.27}$$

$$\text{updated covariance:} \quad \boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \hat{\boldsymbol{P}}_k \tag{3.28}$$

In the places where the state transition matrix $\boldsymbol{F}_k$ and observation matrix $\boldsymbol{H}_k$ can not be replaced by the corresponding functions, they are defined by their derivative:

$$\boldsymbol{F}_k = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{k-1}, \boldsymbol{u}_k} \tag{3.29}$$

$$\boldsymbol{H}_k = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}_k}} \tag{3.30}$$

### 3.3.2 Non-additive Noise EKF

Even though (3.20, 3.21) no longer require a linear relationship for $\boldsymbol{x}_{k-1}$ and $\boldsymbol{u}_k$, the noise has to be additive. We now consider the following formulation, where this no longer is the case:

$$\boldsymbol{x}_k = \boldsymbol{f}\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k, \boldsymbol{w}_k\right) \tag{3.31}$$

$$\boldsymbol{z}_k = \boldsymbol{h}\left(\boldsymbol{x}_k, \boldsymbol{v}_k\right) \tag{3.32}$$

We still assume the noise $\boldsymbol{w}_k$ and $\boldsymbol{v}_k$ to be zero mean white Gaussian noise with their respective covariances $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$, but no longer additive. The formulation for the prediction and update step stay almost the same, only the usage of $\boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ in (3.23, 3.25) are changed slightly:

$$\text{predicted covariance:} \qquad \hat{\boldsymbol{P}}_k = \boldsymbol{F}_k \boldsymbol{P}_{k-1} \boldsymbol{F}_k^\top + \boldsymbol{L}_k \boldsymbol{Q}_k \boldsymbol{L}_k^\top \tag{3.33}$$

$$\text{Innovation covariance:} \qquad \boldsymbol{S}_k = \boldsymbol{H}_k \hat{\boldsymbol{P}}_k \boldsymbol{H}_k^\top + \boldsymbol{M}_k \boldsymbol{R}_k \boldsymbol{M}_k^\top \tag{3.34}$$

with $\boldsymbol{L}_k$ and $\boldsymbol{M}_k$ defined by:

$$\boldsymbol{L}_k = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}\right|_{\boldsymbol{x}_{k-1}, \boldsymbol{u}_k} \tag{3.35}$$

$$\boldsymbol{M}_k = \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{v}}\right|_{\hat{\boldsymbol{x}_k}} \tag{3.36}$$

This not only allows the noise to be non-additive, but also in particular for the process noise, there no longer has to be one noise element for each dimension of the state, so the process noise can now more accurately represent the actual uncertainties within the state estimation. [46]

### 3.3.3 Missing Observations

So far we assumed that for each time step in which the Kalman Filter is executed, there also exists new observational data. But this is not necessarily the case, and [47] introduces a way to handle missing the observations. If $m$ represents the maximum number of observations, so the dimension of $\boldsymbol{z}$, we now define $m_k$ to be the actual number of new observation at time step $k$. This can be as low as 0 or at most $m$. The elements of $\boldsymbol{z}_k$, that are available are at index $i_1, i_2, \ldots, i_{m_k}$ with $1 \leq i_1 \leq i_2 \leq \cdots \leq i_{m_k} \leq m$. From this we create a matrix $\boldsymbol{G}_k$ of dimension $m_k \times m$, with ones at the entries $(1, i_1), (2, i_2), \ldots, (m, m_k)$ and zeros everywhere else. This matrix represents the observations that actually exist at time step $k$. We use it to change the update step of the EKF to the following:

$$\text{Innovation:} \qquad \boldsymbol{y}_k^* = \boldsymbol{z}_k - \boldsymbol{G}_k \boldsymbol{h}\left(\hat{\boldsymbol{x}_k}\right) \tag{3.37}$$

$$\text{Innovation covariance:} \qquad \boldsymbol{S}_k^* = \boldsymbol{H}_k^* \hat{\boldsymbol{P}}_k \boldsymbol{H}_k^{*\top} + \boldsymbol{G}_k \boldsymbol{M}_k \boldsymbol{R}_k \boldsymbol{M}_k^\top \boldsymbol{G}_k^\top \tag{3.38}$$

$$\text{Kalman Gain:} \qquad \boldsymbol{K}_k^* = \hat{\boldsymbol{P}}_k \boldsymbol{H}_k^{*\top} \boldsymbol{S}_k^{*-1} \tag{3.39}$$

$$\text{updated state:} \qquad \boldsymbol{x}_k = \hat{\boldsymbol{x}_k} + \boldsymbol{K}_k^* \boldsymbol{y}_k^* \tag{3.40}$$

$$\text{updated covariance:} \qquad \boldsymbol{P}_k = \left(\boldsymbol{I} - \boldsymbol{K}_k^* \boldsymbol{H}_k^*\right) \hat{\boldsymbol{P}}_k \tag{3.41}$$

where $\boldsymbol{H}_k^* = \boldsymbol{G}_k \boldsymbol{H}_k$. After the prediction step of the EKF, we determine $m_k$, set up $\boldsymbol{G}_k$ accordingly and then preform the adapted update step. In the case that $m_k$ is 0, the update step will not change the prediction at all, meaning it is skipped entirely.

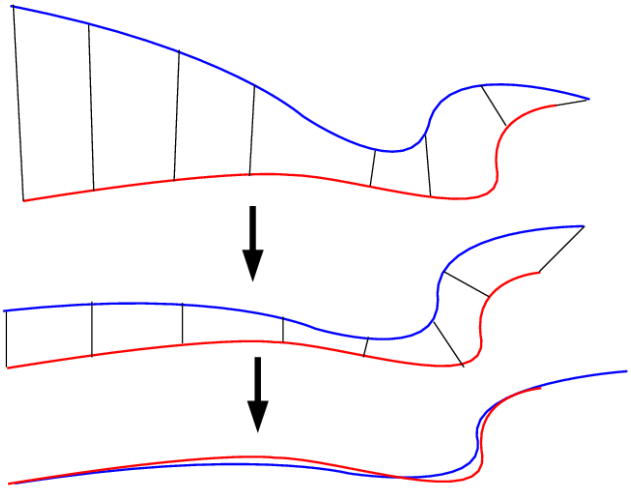


**Figure 3.2:** A 2D illustration of the ICP algorithm, in which the blue data set is mapped onto the red model set. [48]

## 3.4 Iterative Closest Point

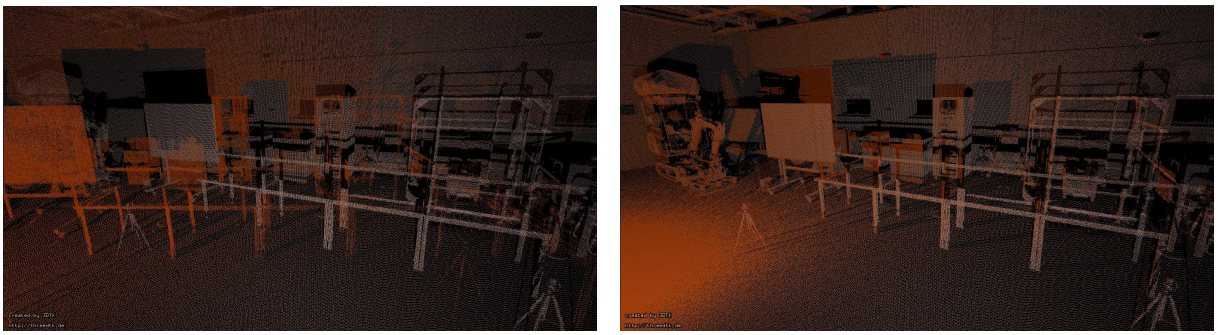
To evaluate the performance of our UWB localization system, we need a ground truth. This is created in part by using laser scans to create multiple point clouds from varying positions, which have to be aligned to determine their relative pose and with that the robots pose to one another. This alignment process is done using the ICP algorithm, which iteratively matches one 2D or 3D point cloud (the data set) to another (the model set). [41, 50–53] A simple 2D illustration of the process is depicted in Figure 3.2, whereas Figure 3.3 shows the performance of the algorithm with 3D point clouds used for our experiments, in which the orange data set laser scan is mapped onto the white model set. In each iteration of the algorithm the following steps are taken:

1. Find the closest point in the model set $\boldsymbol{m}_i$ for each point in the data set $\boldsymbol{d}_i$.
2. Compute the optimum rotation $\boldsymbol{R}$ and translation $\boldsymbol{t}$, that minimize the mean-square error function (3.42) [41], where $N$ is the total number of points in the data set.

$$\frac{1}{N}\sum_{i=1}^{N}\|\boldsymbol{m}_i - (\boldsymbol{R}\boldsymbol{d}_i + \boldsymbol{t})\|^2 \tag{3.42}$$

3. Apply the optimal rotation and translation found in Step 2 to all points in the data set.
4. Stop iterating, if the result of the mean-square error function (3.42) is smaller than a set threshold value.

It is shown in [41], that the ICP algorithm finds a local minimum for the alignment of the point clouds, where the combination of all the rotations $\boldsymbol{R}$ and translations $\boldsymbol{t}$ of the iterations represent the relative pose of the two scans to one another. From this data we extract the pose of the



**(a)** First frame of ICP **(b)** Last frame of ICP

**Figure 3.3:** First and last frames of an example alignment using the ICP algorithm. The white point cloud is the model set, the orange the data set. The two point clouds are taken in the Robotics Hall at the University Würzburg. Visualization created with [49]

robot at the positions where a laser scan is taken with a laser scanner mounted on the robot. The exact transformation from the relative position of the laser scans to the pose of the robot described in Chapter 4.3.2.

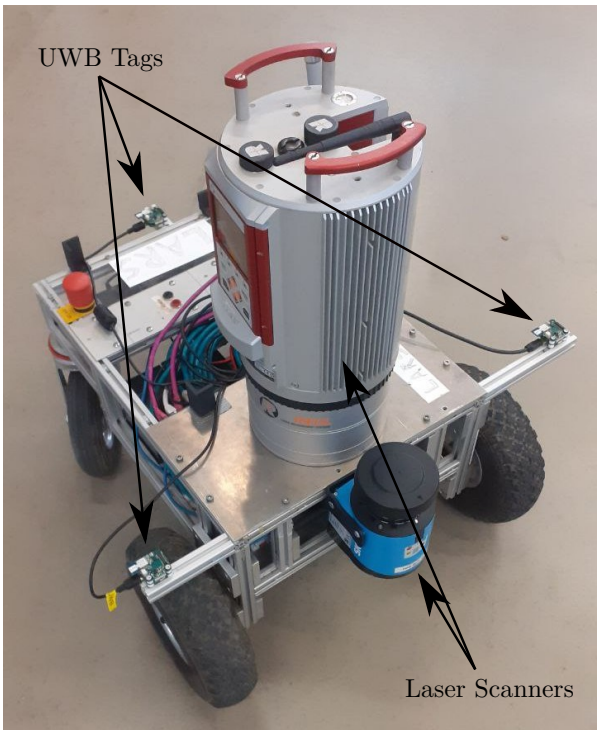
# Chapter 4

# Approach

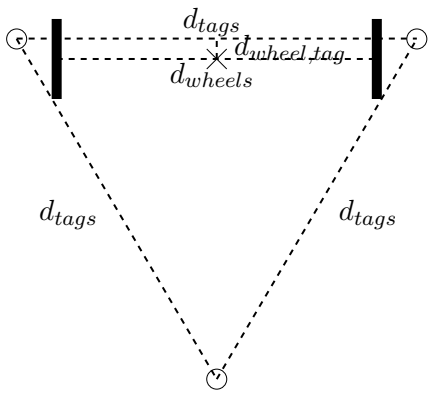
In this chapter we describe the approach we take to preform localization with UWB sensors, and how we evaluate the results. We first discuss the mobile robot used as a test platform, and the relevant equipment mounted on it to support the testing. Then the actual implementation of the TWR scheme and the analysis of that data, to preform the UWB localization. Finally, we explain the different types of systems we use to define a ground truth, to compare the UWB localization to.

## 4.1 Mobile Robot

The robotic system used to test the UWB localization is a four-wheeled mobile robot based on a Volksbot RT3 [54]. It has two independently driven parallel front wheels spaced $d_{wheels} = 44\,\text{cm}$ apart and two castor wheels at the back to support the robots structure. Therefore, it is classified as a differential drive mobile robot with a degree of maneuverability of 2, meaning it is able to drive forwards and backwards, turn and rotate on the spot. The two front wheels are each powered by a 150 W Maxon DC motor through a $74:1$ planetary gearbox. The motors are equipped digital encoders, providing wheel odometry. A specially designed motor controller for the Volksbot platform powers the motors and offers a serial interface for communication. The PC running the robot's software (in our case my personal laptop) connects with the motor controller through this interface, to drive the motors and gather wheel odometry from them. The software running the robot is lightweight C++ code based on ROS [6] connecting to all attached hardware either via Universal Serial Bus (USB) or Ethernet [55]. This includes a Logitech Wireless Gamepad F710 [56], which in our experiments is the exclusive way to drive the robot. The main frame of the robot is made of 20 mm aluminum extrusion, ideal to adapt the robot to whatever requirements necessary and mount additional hardware to. The main payload of the robot are three custom UWB PCBs, mounted to the frame at known positions and connected to the PC. These tags of the UWB network are mounted in an equilateral triangle with side length $d_{tags} = 60\,\text{cm}$. Two of the tags are mounted in the front parallel to the wheel axis, with a distance of $d_{wheel,tag} = 1.6\,\text{cm}$. Figure 4.1b gives a schematic overview of the positioning of the UWB sensors relative to the wheels. Besides this there are two laser scanners on the robot, to provide a ground truth. A SICK LMS100-10000 [57] is mounted of the front of the robot.




**(a)** A picture of the robot, with the laser scanners and tags pointed out.




**(b)** A schematic of the robots driven wheels and tags. The cross represents the zero point of the coordinate frame of the robot, which is directly in between the driven wheels, the circles represent the positions of the UWB tags.

**Figure 4.1:** The robotic system used as a platform to test the UWB localization system.

This is a 2D class 1 Laser Scanner, which operates at 905 nm, has a 270 ° aperture angle and up to 20 m of range. It provides us with quasi continuous coverage of the plane parallel to and ca. 30 cm above the ground. Mounted on top of the robot is a RIEGL VZ-400 Laser Scanner [58]. This near infrared 3D class 1 Laser Scanner has a 360 ° horizontal and a 100 ° (-40 °/+60 °) vertical scan range. The resolution of the laser scans are heavily adjustable, potentially as low as 1.8 arcsec. This laser scanner provides non-continuous high definition 3D laser scans, which at fixed points, where the robot is at a standstill, provides us with a ground truth for both the robot and the UWB anchors. The laser scanner is mounted 13.5 cm behind the zero point of the robot (the cross in Figure 4.1b) and 43.25 cm above it. It is also rotated by 120 ° counter-clockwise. Figure 4.1a is a photo of the robot, in which the two laser scanners and the three UWB tags are pointed out.

## 4.2 UWB Localization

Figure 4.2 outlines a high level overview of the UWB localization process. The UWB nodes run completely independently of the evaluation of their data. They continuously collect distance measurements in between each other and send that for processing to the PC on the robot. The distances between the tags and anchors get used to determine the position of the anchor using

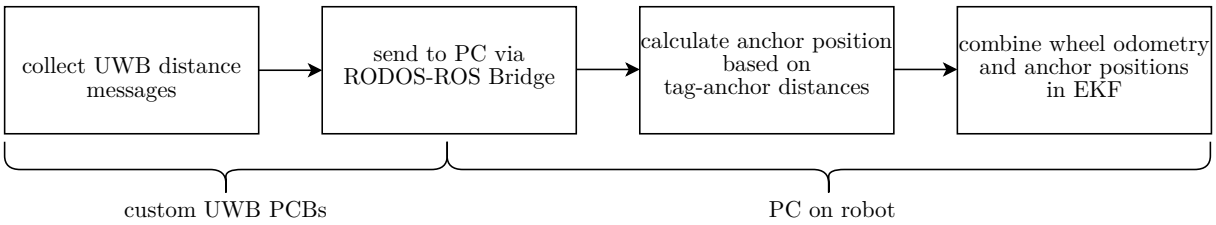



**Figure 4.2:** Flowchart of the individual steps used in our UWB localization system.

**(a)** UWB PCB mounted in a stack with a USB communication board on the robot.

**(b)** The front of the PCB.

**(c)** The back of the PCB.

**Figure 4.3:** The custom PCB with the UWB chip.

an error minimization approach, based on the known position of the tags relative to the robot. This as well as the wheel odometry and optionally the measured distances between the anchors get fed into an EKF, which estimates the position of the robot and the distributed anchors.

### 4.2.1 Hardware

We use the DecaWave (now Qorvo) DWM1000 UWB transceiver [4], which implements the design guidelines stated in [10]. For our experiments we use the Channel Nr. 1 of the DWM1000 chip, which corresponds to a center frequency of 3494.4 MHz with a bandwidth of 499.2 MHz. We also set the antenna delay on each of the UWB boards to the results of our experiments in Chapter 5.2. The DWM1000 chip is integrated on a custom designed PCB, which was originally designed as a stackable PCB for a small Unmanned Aerial Vehicle (UAV), that uses among others UWB transceivers for positioning [59]. Each of the PCBs in the stack perform a certain function, are 30.5 mm by 30.5 mm large with a height of 1.6 mm, and are all connected through a 50 pin electrical connector. They all stack with a 5 mm gap, and have three aligned mounting holes, with which the entire stack is safely mounted and through which the gap in between the boards is assured. Two of the boards are used in our setup, the first is the PCB, that integrates the UWB transceiver onto a board with a STM32F407 microprocessor [60]. This microprocessor communicates with the DWM1000 chip via an SPI interface and runs the embedded code to

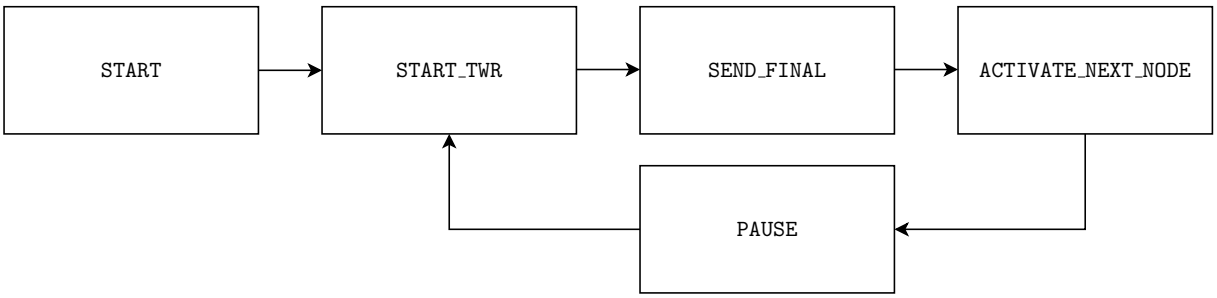



**Figure 4.4:** Overview of the states of the UWB nodes and the transitions in between them.

preform and evaluate the TWR events. The anchors consist of only this board, which is powered using a 9 V block battery and internally also powers the DWM1000 chip. The tags have a second board stacked on the UWB board, which alongside other components has a USB-serial connection, which is used to connect the entire stack with the PC on the robot and also powers the stack through this connection. Figure 4.3 displays the UWB PCB and how they are mounted on the robot.

### 4.2.2 Embedded Software on the UWB nodes

The embedded C++ code running on the STM32F407 microprocessor on the UWB nodes consists of two parts. DecaWave provides source code for relevant Application Programming Interface (API) functions, to interface with the DWM1000 chipset. This includes functions to set general parameters like the channel and antenna delay, read out the tx- and rx-timestamps and send and receive UWB messages, including a delayed transmission option, that allows the transmit time to be set manually. The second part of the embedded code uses these functions to preform the TWR process. This part of the code is based on the specialized operating system RODOS [5]. It is a thread based operating system designed to run on these kinds of embedded development environments on microprocessors, where real time operations play a crucial role. In our case each UWB node runs only a single thread containing a timed infinite loop, which preforms certain actions based on the state the node is in. Figure 4.4 gives an overview of the different states of each of the nodes and how they transition in between them. To understand the individual states and how they relate to the implementation of the TWR scheme, we must first understand the algorithm preformed by each node.

Each of the nodes has a unique hardware based serial number, which is used to order the nodes. Before the program is written on to the boards, we must define a unique `nodeIndex` for each of the nodes, which are integers running from 0 upwards. It is imperative that the node with `nodeIndex` 0 is a tag, otherwise it is only necessary, that the nodes in use are the ones with the smallest possible number. So we are able to use as many or few nodes as we like, we only have to choose those, with serial numbers corresponding to low indices, so that we do not have to reprogram the used nodes. The `nodeIndex` is used as a general proxy to address the nodes in the UWB messages. Each message consists of at least ten bytes, four of which are constant, three are set by the DecaWave API and the remaining three or more bytes are set by us. These bytes are one byte each for the source and the destination address of the node and one byte

| Message Type | # of Data Bytes | Data Content | Use Case |
|---|---|---|---|
| poll | 0 | - | sent as part of the TWR process |
| response | 0 | - | sent back, after receiving the poll message |
| final | $5 \times (2 + n)$ | following timestamps each represented by five bytes: $T_{p,tx}, T_{f,tx}$ and $T_{r,rx}$ for each of the nodes that responded | final message of the TWR process, including the timestamps, that need to be shared |
| distance | $4 + 1$ | distance measurement (`float32_t`), `nodeIndex` of the node, which started the TWR process, the other involved node is the one, which sends the message | reports the calculated distance based on the timestamps collected during the TWR process |
| activate | 1 | $n$ | activates another node, which will then transition into the next state |
| state | 1 | `state` of the node | reports the current state of the node |
| reset | 0 | - | resets the node this is sent to |

**Table 4.1:** List of the used UWB message types, the message type specific data they transmit and how they are used in our implementation of continuous TWR measurements.

for the type of message sent. Messages of certain types have additional bytes, that are used to transmit other data like timestamps. Table 4.1 gives an overview of all the messages used in our implementation of the TWR process, their respective size, additional data bytes and what that data represents and their use case. These messages are then used in our implementation of the TWR scheme. As seen in Table 4.1 the final message is unique, as it is the only one with a variable amount of data, depending on the total amount of active nodes $n$. This is because we preform the TWR scheme not just with one responding node at a time, but with all. So the poll message is sent out not to one specific other node but rather to all, this is noted by the fact, that the destination byte in the message is set to the maximum possible one byte value 255. The response messages sent back from the varying nodes, are staggered so that they do not overlap, which means only one message has to be received at a time. This follows the implementation for concurrent ranging outlined in [19]. There are other options to preform concurrent ranging with multiple responders like those introduced in [61–63], but these rely at least in part on the possibility to receive multiple messages at the same time, which is not supported by the API

from DecaWave. Therefore, we use an implementation which is based on staggered response messages, meaning the responders have to wait a certain amount of time before sending the response message. This leads to large difference in the reply time of the various nodes, which makes SDS-TWR completely useless, as the timing error due to the clock drift depends on the difference of the reply times (cf. (3.5)). Therefore, we use ADS-TWR. The wait time before responding to a message is defined by the `nodeIndex` of the corresponding node, thus insuring a staggered response. The variable amount of data in the final message represents the $T_{r,rx}$ timestamps from each responding node and the $T_{p,tx}$ and $T_{f,tx}$ timestamps. We are able to send the $T_{f,tx}$ timestamp along with the final message, even though at the time it has to be written into the relevant register the message is not sent yet, as we use the timed transmission feature of the DWM1000 chip for this message.

**Algorithm 1:** The general principle of our implementation of the TWR scheme. This does not represent the actual implementation of an UWB node.

```
i ← 0
while true do
    node i sends poll msg
    forall j ∈ {0, 1, ..., n − 1} \ i do
        node j sends response msg after j × respWaitTimePerNode
    end
    node i waits n × respWaitTimePerNode after it sent poll msg
    node i populates and sends final msg
    forall j ∈ {0, 1, ..., n − 1} \ i do
        node j calculates distance
        if node j is tag then
            node j publishes distance to the RODOS-ROS-Bridge
        else
            node j sends distance via distance msg to node 0
        end
    end
    node 0 publishes all distance values received to the RODOS-ROS-Bridge
    node i waits n × respWaitTimePerNode after it sent final msg
    node i sends activate msg to node ((i + 1) mod n)
    i ← ((i + 1) mod n)
end
```

Algorithm 1 gives a general overview of our implementation of the ADS-TWR process, whereas Algorithm 2 represents the actual implementation of a node. In the beginning all the nodes are in the `START` state. Here they do nothing except respond to incoming messages. To start the TWR we must once activate the node with index 0. We do this by sending a message over a ROS topic from the PC on the robot. Over the RODOS-ROS-Bridge this message (not a UWB message) is transferred to all connected UWB nodes. These are all the tags, that are permanently connected to the robot. The only node, that checks the ROS messages coming in from the RODOS-ROS-Bridge is the node with index 0, which we ensure to be a tag. Therefore,

**Algorithm 2:** An overview of the implementation for a specific *node*.

```
nextTime2Action ← NOW()
while true do
    if nextTime2Action ≤ NOW() then
        switch node.state do
            case START_TWR do
                sendPoll() and set T_{p,tx} timestamp
                node.toggleState()
                nextTime2Action ← NOW() + n × respWaitTimePerNode
            end
            case SEND_FINAL do
                set T_{f,tx} timestamp and sendFinal() at T_{f,tx}
                node.toggleState()
                nextTime2Action ← NOW() + n × respWaitTimePerNode
            end
            case ACTIVATE_NEXT_NODE do
                sendActivate((nodeIndex + 1) mod n)
                node.toggleState()
            end
        end
    end
    if new msg received then
        switch msg.type do
            case poll do
                set T_{p,rx} timestamp
                waitUntil(NOW() + nodeIndex × respWaitTimePerNode)
                sendResponse() and set T_{r,tx} timestamp
            end
            case response do
                set T_{r,rx} timestamp
            end
            case final do
                set T_{f,rx} timestamp
                T_{p,tx}, T_{r,rx}, T_{f,tx} ← msg.data
                calculate from timestamps and publish distance to RODOS-ROS-Bridge
            end
            case activate do
                node.toggleState()
                n ← msg.data
                nextTime2Action ← NOW()
            end
        end
    end
end
```

we guarantee that only one node gets put into the `START_TWR` state. Within this message we send how many nodes are active overall $n$, which gets saved in the variable `numberOfActiveNodes`, this enables us to use as many nodes as we like without having to update the embedded source code on the nodes. Starting now the UWB nodes run completely independently of the PC running ROS, only reporting the measured distances to it. One node after the other will start a TWR process. For the following explanation of the algorithm used, we assume node $i$ is the one, that starts the TWR process. Node $i$ starts the TWR process by sending out a poll message, saving the $T_{p,tx}$ timestamp and transitioning into the `SEND_FINAL` state. All other active nodes, no matter what state they are in, receive this message, set the $T_{p,rx}$ timestamp on their node, wait some time, corresponding to their `nodeIndex`, send the response message back to node 0 and finally set the $T_{r,tx}$ timestamp. The wait time of each node is their `nodeIndex` times `respWaitTimePerNode`, which is 5 ms (cf. Chapter 5.3). Node $i$ receives all response messages and sets the corresponding timestamps $T_{r,rx}$ for each of the responding nodes. As the `nodeIndex` of the nodes are zero-indexed and as small as possible, the largest `nodeIndex` is $n-1$. So at most 5 ms times $n$ after the poll message is sent out all response messages are received. After this wait time node $i$ preforms the next action, this time sending out the final message based on its state. For this we set the transmission time 3 ms into the future to allow some time to populate the data part of the final message. The $T_{f,tx}$ timestamp is set to the transmission time plus the antenna delay, for the other timestamps the DecaWave API automatically adds or subtracts the antenna delay respectively. We enter all timestamps collected by node $i$ into the final message, before it is sent using the timed transmission feature of the DWM1000 chip and node $i$ transitions into the `ACTIVATE_NEXT_NODE` state. The other active nodes receive the final message, timestamp its reception $T_{f,rx}$ and read out the data from the message. With the two tx-timestamps from node $i$, the $T_{r,rx}$ timestamp corresponding to their node and the three timestamp they recorded themselves they calculate the distance between them and node $i$ using (3.6). Now comes the only implementation difference between the tags and the anchor: As the tags are connected directly to the PC on the robot via the RODOS-ROS-Bridge, they publish the calculated distance along with the two node indices, in between which the distance is measured, directly onto that, whereas the anchors first send it using a UWB distance message to node 0, which is required to be a tag, which then publishes it via the RODOS-ROS-Bridge. The distance messages sent to node 0 by the anchors also have to be staggered, so that they do not overlap, this is done in the same way as with the response messages. Another at most 5 ms times $n$ after the final message is sent out all distances are reported, so that is the time for the next action of node $i$. Now the node activates the next node transitioning it into the `START_TWR` state coming from either the `PAUSE` or `START` state and then transitions itself into the `PAUSE` state (cf. Figure 4.4). The next node is just the node with an index one larger than node $i$ modulo $n$ ($i \leftarrow (i+1 \mod n)$). From there the TWR process starts again, now from the next node, which will continuously activate one node after the other. After a full cycle consisting of $n$ TWR measurement processes one from each node assuming no messages were lost we have two distance measurements for each of the $\binom{n}{2}$ combination of nodes one from either side. This cycle gets repeated indefinitely.

This implementation is set up to measure the distance to all other active nodes at once during one TWR execution, instead of just one. An implementation only preforming a single distance measurement per TWR execution, is significantly easier, as there is no longer the requirement

to stagger any messages nor wait for multiple responses. In that case both the response and distance message are sent out immediately after the poll or final message is received and also sending out the final message is done as soon as the response message is received. Also, the data structure of final message is simpler as it does not require space for a variable amount of $T_{r,rx}$ timestamps, instead just one. The advantage of the additional implementation overhead to preform multiple distance measurements at once is on average fewer UWB messages sent per distance measurement, and therefore a higher update rate. Considering one TWR execution with concurrent ranging $1 + (n - 1) + 1$ UWB messages are needed to preform the ADS-TWR process. To preform $n - 1$ TWR executions without concurrent ranging to measure the same distances, $3(n - 1)$ messages are needed. Therefore, we save $2n - 2$ messages, corresponding in most of our experiments, where we use three tags and four anchors, so $n = 7$, to a decrease of over 50 % for the core ADS-TWR process.

There are multiple potential issues that arise, if a message is not properly transmitted, so we have implemented safeguards, that ensure the automatic cycling of TWR measurements does not fail, if a message is not properly transmitted. If a final or distance message is lost, the corresponding distance measurement is also lost in this cycle, but it does not have an effect on the continued cycle. The sending node will simply activate the next node after $5 \times n$ ms, just as it does anyway. If a poll or response message is lost it has an impact on the distance calculation, because the corresponding timestamps are missing, or still have the values from the previous iteration. To mitigate this all $T_{r,rx}$ are set to the maximum 64-bit value, when a poll message is sent. As the timestamps are only 40-bit values, but are saved in 64-bit variables, a correct timestamp will never be the maximum 64-bit value. If either the poll or response message fail, no response is registered, so no correct $T_{r,rx}$ timestamp is saved. From this we ensure, that the distance calculation and publication for the node corresponding to the failed poll or response message is skipped in this iteration. In doing so we save a wrong distance from being published, although as the node sends out the final and activate messages automatically, each after $5 \times n$ ms, this does not have an impact on the continued cycle either. This leaves us with the activate message. If this is not received, it breaks the entire cycle, because all nodes are then in the `PAUSE` state, in which they will stay indefinitely. To prevent this the node, which sends the response message, waits for a message to come in within the next 50 ms. If the next node correctly receives the activate message, it sends a confirmation message back. The poll message of the then starting TWR process, also reaches the original node as a confirmation within 50 ms. If no message is received in the 50 ms timeframe, the node sends out another activate message, but to one node further (so `nodeIndex` $+ 2 \mod n$). This ensures that if a node is not reachable either temporarily or indefinitely, the cycle still continues. The possibility that the activate message is received, but both the confirmation and poll message is not received is ignored, as it is implausible, that two successive messages fail over the same link, that immediately before that successfully transmitted the activate message. If a node does not activate any of the other nodes it stops and all nodes fall into the `PAUSE` state. The other way this happens, is if the node that is currently preforming the TWR process dies, before activating the next node. For this special circumstance the node 0, which is a tag and therefore connected to and powered through the PC on the robot, toggles itself into the `START_TWR` state, if it has been in the `PAUSE` state for at least 3 s. This is plenty of time for a successful cycle of TWR measurements, so it allows us to have stable, continuously running, fully self regulated distance measurements. It

also allows an anchor to fail and then reappear, because it e.g. is hidden behind a rock from a certain perspective, and still integrate seamlessly back into the measurement cycle. Besides the slight difference in the way the distance measurements are reported, this and the one initial start activation from a ROS message is the only difference between the embedded code for the tags and anchors.

### 4.2.3 Anchor Position Determination

The evaluation code running on the PC on the robot is split into two ROS nodes. First the gathered distance measurements from the UWB sensors are further processed into positional data of the anchors. The second node (cf. Chapter 4.2.4) is responsible for running the EKF, which uses the positional data along with wheel odometry as an input. Once the ROS node for the anchor position determination is started, it publishes one activate message with the number of used UWB nodes, which is set manually, to the RODOS-ROS-Bridge. After that it continuously scans for new distance messages from the RODOS-ROS-Bridge and records the distances, as well as the node indices in between which the distance was measured. As soon as it detects a full cycle of $n$ TWR processes is done, it calculates the position of the anchors based on the distance measurements and the geometry of the tags shown in Figure 4.1b. This is done independently for each of the anchors. For each anchor it is first checked, that all necessary distance measurements were taken in the last cycle, if not the position determination for this anchor is skipped in this round. The necessary measurements are the distances from each of the three tags to the anchor and the other way around. The two measurements per tag-anchor combination are averaged $d_{t,a}$ and then used in the following minimization problem:

$$\begin{aligned} &\forall a \in anchors: \\ &\min_{x_a, y_a} \sum_{\forall t \in tags} ((x_a - x_t)^2 + (y_a - y_t)^2 - d_{t,a}^2)^2 \end{aligned} \tag{4.1}$$

$x_a, y_a, x_t$ and $y_t$ represent the $x$- and $y$-position of the corresponding anchor or tag in the coordinate frame of the robot respectively. The coordinate frame of the robot is defined, with the $x$-axis pointing forward (perpendicular to the axis of the driven wheels), and the $y$-axis to the left (along the axis of the driven wheels). The zero point is directly in between the two driven wheels, which in combination with the geometry shown in Figure 4.1b, gives us the following positions for the tags:

$$t1 = \left( -\frac{d_{tags}\sqrt{3}}{2} + d_{wheel,tag}, 0 \right) \tag{4.2}$$

$$t2 = \left( d_{wheel,tag}, -\frac{d_{tags}}{2} \right) \tag{4.3}$$

$$t3 = \left( d_{wheel,tag}, \frac{d_{tags}}{2} \right) \tag{4.4}$$

An example of the minimization problem (4.1) and its solution is depicted in Figure 4.5. With this we have an estimation for the positions of the anchors in the coordinate frame of the robot,

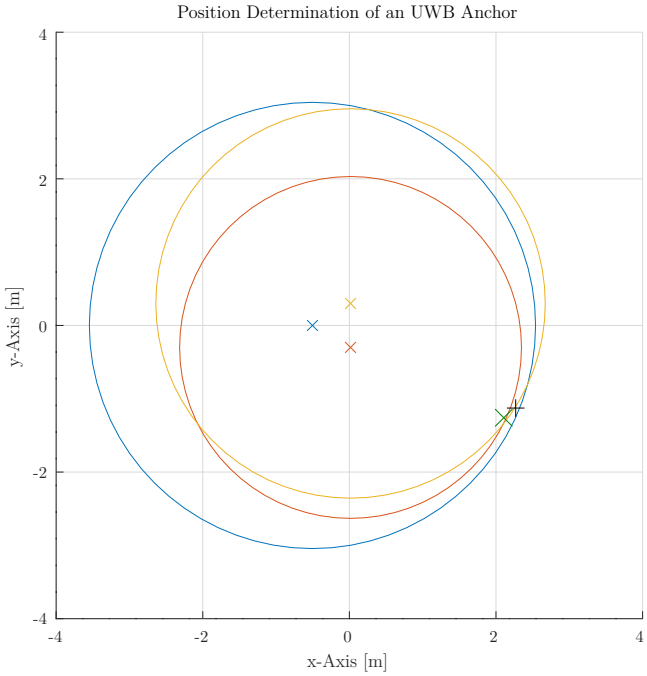



**Figure 4.5:** An examplary depiction of the anchor postion determination process. The blue, orange and yellow crosses (×) and the surrounding circles, represent the position of the tags, with the robot at the origin and their respective distance measurements to the anchor. The black plus (+) is the determined position of the anchor based on (4.1). The green cross (×) depicts the true position of the anchor (cf. Chapter 4.3.2).

which is published to a ROS topic to be used by the EKF. After this calculation the distance measurements of the just passed cycle are deleted, to ensure, that if in the new cycle some distance measurements are missing, the outdated measurements are not used again to calculate the anchors position.

### 4.2.4 EKF-SLAM

The positional data of the anchors in addition to the wheel odometry is used in an EKF to estimate the pose of the robot as well as increasing the precision of the anchor position estimations. The EKF we use is inspired by an EKF introduced in [64]. In this paper the authors expand on their previous work from [65], in which they introduce a Multi-Hypothesis EKF, to estimate both the range and the bearing of a RFID tag. Similarly to the UWB nodes, they measure the distance from a robot to the RFID tag and use this along with the wheel odometry of the robot to estimate also the bearing of the RFID tag. This is then used as an inner EKF for each RFID tag in [64], to use as an observation in the outer EKF which preforms a SLAM based algorithm to estimate the pose of the robot and the positions of the RFID tags. In our implementation this is modified, as we have not just a distance measurement from the UWB anchors to the robot, but due to the fact, that we have three tags on the robot, we have full positional information for

each anchor. Therefore, we do not need the inner EKF to estimate the bearing of the anchors. We use an EKF, that also takes into account non-additive noise and the possibility of missing measurements (cf. Chapter 3.3). We consider a system with $n$ anchors in use, in which case the state $\boldsymbol{x}$ of the EKF has $3 + 2n$ entries:

$$\boldsymbol{x} = [x_R, y_R, \theta, x_{A,1}, y_{A,1}, \ldots, x_{A,n}, y_{A,n}] \tag{4.5}$$

$x_R$ and $y_R$ represent the position of the robot, $\theta$ the orientation and $x_{A,i}$ and $y_{A,i}$ the position of the $i$th anchor. All of these coordinates are in the fixed global frame. Using the kinematics of the robot we define $\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$ as:

$$\hat{\boldsymbol{x}_k} = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) = \Big[x_{R,k-1} + v_k \cos\theta_k, y_{R,k-1} + v_k \sin\theta_k, \theta_k + \omega_k, x_{A,1}, y_{A,1}, \ldots, x_{A,n}, y_{A,n}\Big] \tag{4.6}$$

with $v_k = \frac{v_{r,k}+v_{l,k}}{2}$ and $\omega_k = \frac{v_{r,k}-v_{l,k}}{d_{wheels}}$, where $v_{r,k}$ and $v_{l,k}$ are the distances the wheels moved forward during the last iteration of the EKF, which is the exact wheel odometry data provided by the motor driver. The $x$ and $y$ position of the anchors in (4.6) stay unchanged, as we assume, that after deployment the anchors are stationary. The predicted covariance of the system is calculated as follows:

$$\hat{\boldsymbol{P}}_k = \boldsymbol{F}_k \boldsymbol{P}_{k-1} \boldsymbol{F}_k^\top + \boldsymbol{L}_k \boldsymbol{Q}_k \boldsymbol{L}_k^\top \tag{4.7}$$

With:

$$\boldsymbol{F}_k = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}_{k-1}, \boldsymbol{u}_k} = \begin{bmatrix} 1 & 0 & 0 & 0 & & 0 \\ 0 & 1 & 0 & 0 & & 0 \\ -v_k \sin\theta_k & v_k \cos\theta_k & 1 & 0 & \ldots & 0 \\ 0 & 0 & 0 & 1 & & 0 \\ & & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & 1 \end{bmatrix} \tag{4.8}$$

The matrix $\boldsymbol{Q}$ represents the error $\boldsymbol{w}$ of the wheel odometry, which is $\boldsymbol{Q}_k = \begin{bmatrix} K_r \left|v_{r,k}\right| & 0 \\ 0 & K_l \left|v_{l,k}\right| \end{bmatrix}$ with $K_r$ and $K_l$ constants, that describe the accuracy of the wheel odometry. As these errors are non-additive to the state, $\boldsymbol{L}_k$ is defined as:

$$\boldsymbol{L}_k = \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{w}}\right|_{\boldsymbol{x}_{k-1}, \boldsymbol{u}_k} = \begin{bmatrix} \frac{\cos\theta_k}{2} & \frac{\cos\theta_k}{2} \\ \frac{\sin\theta_k}{2} & \frac{\sin\theta_k}{2} \\ \frac{1}{d_{wheels}} & \frac{1}{d_{wheels}} \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \end{bmatrix} \tag{4.9}$$

For the innovation $\boldsymbol{y}_k^* = \boldsymbol{z}_k - \boldsymbol{G}_k \boldsymbol{h}(\hat{\boldsymbol{x}_k})$, with $\boldsymbol{G}_k$ the matrix encoding the observations, that are not missing, we define $\boldsymbol{h}(\hat{\boldsymbol{x}_k})$. The observations consist of two parts, first the positional

data of the anchors, which is in the robots coordinate frame, and second the inter anchor UWB distance measurements. For these the ROS node, that implements the EKF, also subscribes to the distance topic from the RODOS-ROS-Bridge. Just like with the node, that calculates the position of the anchors based on the tag-anchor distance measurements, this node also ensures, that each anchor-anchor distance measurement is only used once, although it is used as soon as it becomes available, and we do not wait for a full cycle to complete. Therefore, $\boldsymbol{h}\left(\hat{\boldsymbol{x}_k}\right)$ also has to have two parts, first transforming the anchor positions in $\boldsymbol{x}$ into the robot coordinate frame, based on the geometric relationships depicted in Figure 4.6, and second calculating the expected distances between each of the $\binom{n}{2}$ anchor-anchor pairs:

$$\boldsymbol{h}\left(\hat{\boldsymbol{x}}\right) = \begin{bmatrix} \cos\theta\left(x_{\hat{A},1} - \hat{x_R}\right) + \sin\theta\left(y_{\hat{A},1} - \hat{y_R}\right) \\ -\sin\theta\left(x_{\hat{A},1} - \hat{x_R}\right) + \cos\theta\left(y_{\hat{A},1} - \hat{y_R}\right) \\ \cos\theta\left(x_{\hat{A},2} - \hat{x_R}\right) + \sin\theta\left(y_{\hat{A},2} - \hat{y_R}\right) \\ -\sin\theta\left(x_{\hat{A},2} - \hat{x_R}\right) + \cos\theta\left(y_{\hat{A},2} - \hat{y_R}\right) \\ \vdots \\ \cos\theta\left(x_{\hat{A},n} - \hat{x_R}\right) + \sin\theta\left(y_{\hat{A},n} - \hat{y_R}\right) \\ -\sin\theta\left(x_{\hat{A},n} - \hat{x_R}\right) + \cos\theta\left(y_{\hat{A},n} - \hat{y_R}\right) \\ \sqrt{\left(x_{\hat{A},1} - x_{\hat{A},2}\right)^2 + \left(y_{\hat{A},1} - y_{\hat{A},2}\right)^2} \\ \vdots \\ \sqrt{\left(x_{\hat{A},1} - x_{\hat{A},n}\right)^2 + \left(y_{\hat{A},1} - y_{\hat{A},n}\right)^2} \\ \sqrt{\left(x_{\hat{A},2} - x_{\hat{A},3}\right)^2 + \left(y_{\hat{A},2} - y_{\hat{A},3}\right)^2} \\ \vdots \\ \sqrt{\left(x_{A,\hat{n-1}} - x_{\hat{A},n}\right)^2 + \left(y_{A,\hat{n-1}} - y_{\hat{A},n}\right)^2} \end{bmatrix} \tag{4.10}$$

To now calculate the covariance of the innovation and the Kalman gain and finally update both

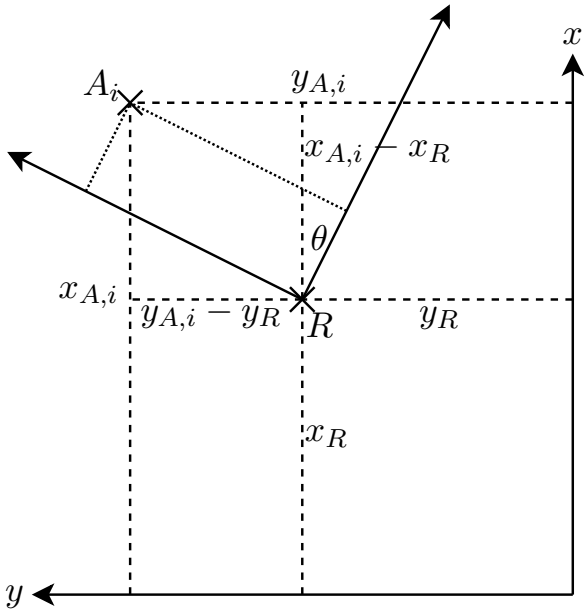



**Figure 4.6:** The robot $R$ at $(x_R, y_R)$ and its coordinate frame at an angle $\theta$ from the global frame, as well as an anchor $A_i$ at $(x_{A,i}, y_{A,i})$ inside the global coordinate frame.

the state and the covariance as per (3.38-3.41) we must determine $\boldsymbol{H}_k = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\Big|_{\hat{\boldsymbol{x}_k}}$:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{A} & \begin{matrix} -(x_{\hat{A},1} - \hat{x_R})\sin\theta + (y_{\hat{A},1} - \hat{y_R})\cos\theta \\ -(x_{\hat{A},1} - \hat{x_R})\cos\theta - (y_{\hat{A},1} - \hat{y_R})\sin\theta \end{matrix} & -\boldsymbol{A} & & \boldsymbol{0}_{2\times 2n-2} & \\ \boldsymbol{A} & \begin{matrix} -(x_{\hat{A},2} - \hat{x_R})\sin\theta + (y_{\hat{A},2} - \hat{y_R})\cos\theta \\ -(x_{\hat{A},2} - \hat{x_R})\cos\theta - (y_{\hat{A},2} - \hat{y_R})\sin\theta \end{matrix} & \boldsymbol{0}_{2\times 2} & -\boldsymbol{A} & \boldsymbol{0}_{2\times 2n-4} & \\ \vdots & \vdots & & & \ddots & \\ \boldsymbol{A} & \begin{matrix} -(x_{\hat{A},n} - \hat{x_R})\sin\theta + (y_{\hat{A},n} - \hat{y_R})\cos\theta \\ -(x_{\hat{A},n} - \hat{x_R})\cos\theta - (y_{\hat{A},n} - \hat{y_R})\sin\theta \end{matrix} & & \boldsymbol{0}_{2\times 2n-2} & & -\boldsymbol{A} \\ & & \boldsymbol{B}_{1,2} & -\boldsymbol{B}_{1,2} & \boldsymbol{0}_{1\times 2n-4} & \\ & & \vdots & & \ddots & \\ & & \boldsymbol{B}_{1,n} & \boldsymbol{0}_{1\times 2n-4} & & -\boldsymbol{B}_{1,n} \\ & \boldsymbol{0}_{\binom{n}{2}\times 3} & \boldsymbol{0}_{1\times 2} & \boldsymbol{B}_{2,3} & -\boldsymbol{B}_{2,3} & \boldsymbol{0}_{1\times 2n-6} \\ & & & & \ddots & \ddots \\ & & & \boldsymbol{0}_{1\times 2n-4} & \boldsymbol{B}_{n-1,n} & -\boldsymbol{B}_{n-1,n} \end{bmatrix} \tag{4.11}$$

$\boldsymbol{A}$ is a shorthand for $\begin{bmatrix} -\cos\theta & -\sin\theta \\ \sin\theta & -\cos\theta \end{bmatrix}$.
$\boldsymbol{B}_{i,j}$ is a shorthand for $\begin{bmatrix} \frac{x_{\hat{A},i}-x_{\hat{A},j}}{\sqrt{\left(x_{\hat{A},i}-x_{\hat{A},j}\right)^2+\left(y_{\hat{A},i}-y_{\hat{A},j}\right)^2}} & \frac{y_{\hat{A},i}-y_{\hat{A},j}}{\sqrt{\left(x_{\hat{A},i}-x_{\hat{A},j}\right)^2+\left(y_{\hat{A},i}-y_{\hat{A},j}\right)^2}} \end{bmatrix}$. We also need to define the matrix $\boldsymbol{R}_k$, which represents the variance of the observations.

$$\boldsymbol{R}_k = \begin{bmatrix} \sigma^2_{anPos}\boldsymbol{I}_{2n\times 2n} & \boldsymbol{0}_{2n\times\binom{n}{2}} \\ \boldsymbol{0}_{\binom{n}{2}\times 2n} & \sigma^2_{dis}\boldsymbol{I}_{\binom{n}{2}\times\binom{n}{2}} \end{bmatrix} \tag{4.12}$$

We define the variances $\sigma^2_{anPos}$ and $\sigma^2_{dis}$ as constants, which we assume to be additive to the $x$ or $y$ position estimation of the anchors and the UWB distance measurements respectively. Therefore, the matrix $\boldsymbol{M}$ from (3.38) is not needed, as it handles a potential non-additive error behavior.

With this the EKF is fully set up, missing is only the initial values for the state $x$ and the covariance $P$. We define the initial position of the robot to be aligned with the global coordinate frame, which also means the covariance of it is 0. For the first estimates of the anchors we wait for a certain amount of measurements for each of the anchor positions to come in. We then use the average over those measurements as the initial estimation for each of the anchor positions and the statistical variance to fill the diagonal entries of $P$ corresponding to the anchor positions. Then each of the steps of the EKF is repeated indefinitely.

In the setup of the EKF we ignored the fact, that the heading of the robot, which is a part of the EKFs state, typically is not represented in a continuous data range. Usually the heading is constrained to the interval $(-\pi, \pi]$, which leads to a discontinuity at $\pm\pi$. To circumvent this [66, 67] propose to wrap an angle $\gamma$ to $\mod(\gamma + \pi, 2\pi) - \pi$ at any time the angle is compared to another angle, i.e. an observation. [67] shows, that this is equivalent to any two-dimensional rotations, also those that cross the discontinuity. In our implementation we do not implement such wrapping, instead the angles range is not constrained at all. As the heading of the robot is never directly compared to any other angle, only used in trigonometric functions, whose domain is not limited to $(-\pi, \pi]$, the wrapping is not necessary. Of course a heading of e.g. $2\pi$ is still equivalent to one of 0, but it represents, that the robot made a full counter-clockwise rotation.

As discussed in Chapter 2.2 a common problem with SLAM algorithms is loop closing. An example of this is shown in Figure 2.4. Loop closing describes the phenomenon, that if the object preforming the SLAM algorithm, in our case the robot, reenters an area already visited, it has to detect that, to combine the two parts of the map into one. In our case this map updated with the EKF is very limited, only consisting of the positions of the distributed anchors. But as each distance measurement includes a label identifying the associated anchor, it is obvious when the robot reenters an area already scanned. This also pertains to the case, that an anchor temporarily drops out.

## 4.3 Ground Truth

A critical part of the evaluation of our novel UWB localization system is a ground truth. This is necessary to compare the results of the localization to determine how successful the localization

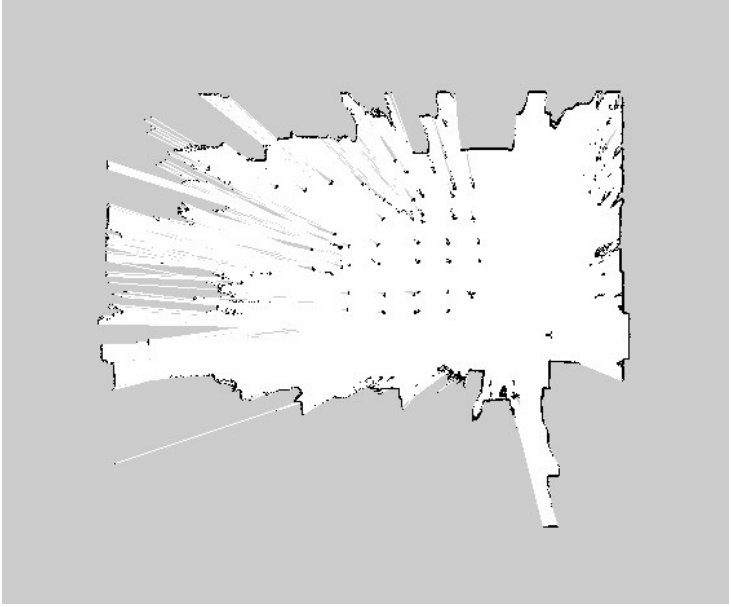


**Figure 4.7:** A map created using the Hector SLAM algorithm. We observe both the outer walls of the Robotics Hall, this scan was taken in, including some open doors and table legs in the center of the room. Created with: [69]

is. For this we use two different system, both based on laser scanners mounted to the robot. The 2D SICK LMS100 laser scanner mounted at the front of the robot, provides us with quasi continuous data, which is used to preform a SLAM algorithm, which provides us with continuous updates to the pose of the robot. Secondly we use the 3D Riegl VZ-400 to gather high definition laser scans, for a more accurate pose of the robot as well as the positions of the anchors.

### 4.3.1 Continuous 2D Laser Scans

We have chosen the Hector SLAM algorithm to use, to continuously determine the pose of the robot [68]. This algorithm is implemented in ROS and runs completely independently of any other ROS nodes [69]. Other than other common SLAM algorithms like tinySLAM [70] or gMapping [71] Hector SLAM only relies on the laser scanner data and does not use e.g. wheel odometry as a second input. The output of the algorithm is a continuous pose of the robot and an updating map of the surroundings, an example of which is shown in Figure 4.7. As we only observe structures like walls and table legs in the map and therefore also in the scan, it is clear, that this kind of localization works the best indoors and has problems in a low feature outdoor scenario.

### 4.3.2 High Definition 3D Laser Scans

The high definition laser scans from the Riegl VZ-400, which is mounted on top of the robot, are used for two purposes. During the course of one experiment run-through we take multiple 3D laser scans, for which the robot has to be and stay at a standstill for the full course of the

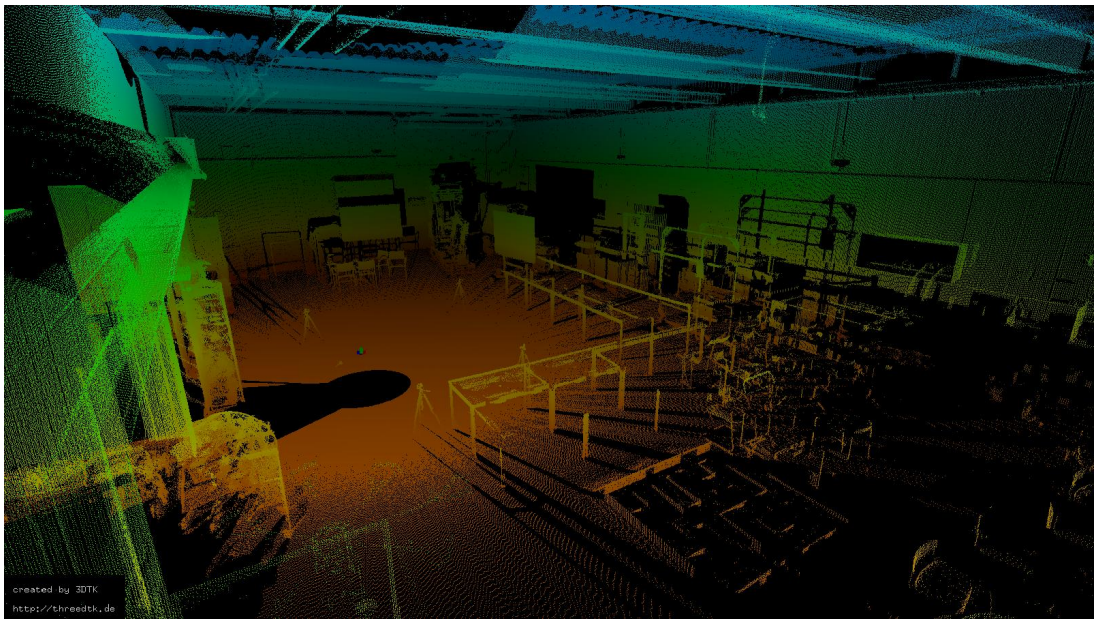



**Figure 4.8:** The point cloud resulting from a 3D laser scan taken with the Riegl VZ-400 of the inside of the Robotics Hall at the University Würzburg, where the UWB localization experiments took place. The color gradient from red to green to blue indicates the height of the individual points. Visualization created with [49]

scan. An example of such a scan is depicted in Figure 4.8. There are three relevant coordinate frames associated with these scans. First the fixed global coordinate frame $O$, second the moving robots coordinate frame $R$ and third the coordinate frame of the laser scanner $L$, which is fixed to the robot, but also changing in the global frame. $T_{Ai,Bk}$ represents the transformation from coordinate frame $A$ at the $i$th laser scan to the frame $B$ at the $k$th scan, though $O$ does not change from one scan to the next. The first use of the scans is the determination of the robots pose at the scan locations. We always take a first scan at the position we start the experiment run, which is set to be the origin of the fixed global coordinate frame. After the experiment we preform the ICP algorithm described in Chapter 3.4 for which we use the implementation from [49]. The output of this are one $4 \times 4$ homogenous transformation matrix $T_{O,Li} \forall i \in \{2, \ldots, n_S\}$ for each of the $n_S$ laser scans, except for the first scan, which is used as the model set for the ICP algorithm. As the robots pose at the first laser scan is in line with the origin of the global coordinate frame $O$, the transformation matrix $T_{O,L1}$ of the first laser scan is equal to the known transformation from the robot to the laser scanner coordinate frame $T_{R1,L1}$. This transformation is defined by the mounting pose of the scanner on the robot described in Chapter 4.1. From which we set:

$$T_{R1,L1} = \begin{bmatrix} -0.5 & -\frac{\sqrt{3}}{2} & 0 & -0.135 \\ \frac{\sqrt{3}}{2} & -0.5 & 0 & 0 \\ 0 & 0 & 1 & 0.4324 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.13}$$

As the laser scanner is permanently fixed to the robot this is the same at the time of all scans $T_{O,L1} = T_{Ri,Li} \, \forall i \in \{1, \ldots, n_S\}$. This fact is used to determine the robots pose at all the scan locations using the output of the ICP algorithm:

$$T_{O,Ri} = T_{O,Li} T_{Li,Ri} = T_{O,Li} T_{R1,L1}^{-1} \tag{4.14}$$

With this we have the pose of the robot at all the locations of the laser scans. Even though this does not enable a continuous comparison for the UWB localization, it does enable a very precise comparison at the laser scan locations not just for the UWB localization but also for the Hector SLAM.

The second use is to determine the positions of the anchors, which is also determined by our implementation of the EKF SLAM, and need a ground truth to compare to. For this we select the anchors in the point cloud of the first laser scan, by clicking on them using a tool provided by [49]. This provides us with the coordinates for each of the $n$ anchors in the coordinate frame of the first laser scan $p_{i,L1} \, \forall i \in \{1, \ldots, n\}$. These points get transformed into the global coordinate frame

$$p_{i,O} = T_{O,L1} p_{i,L1} = T_{R1,L1} p_{i,L1} \, \forall i \in \{1, \ldots, n\} \tag{4.15}$$

and represent the position of the UWB anchors, which are deployed at previously unknown points.

Combined with the Hector SLAM this provides us with a continuous ground truth to compare our UWB localization and anchor mapping to, and a more accurate ground truth for specific points along the driven path.

# Chapter 5

# Experiments and Evaluation

In this chapter we present all the different experiments we have done, their results and how they affected the further system design. In Chapter 5.5 the fully integrated UWB localization system is tested, as it is described in detail in Chapter 4. The first sections of this chapter highlight the experiments done for particular sub-aspects of the full system setup, from first tests of the functional principle of the UWB boards and the TWR process to the antenna delay calibration to timing optimizations for better performance and the anchor position determination.

## 5.1 UWB Functional Principle

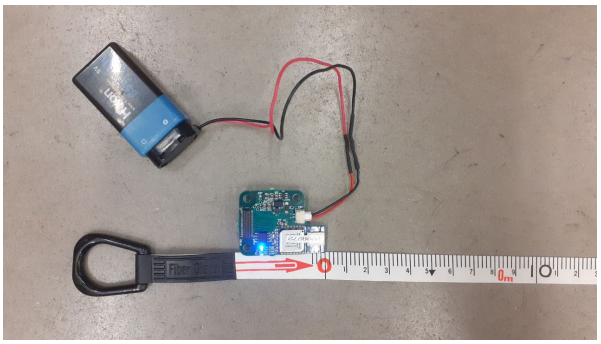
The first experiments we preform are simple distance measurements using two UWB nodes, where just one node preforms the TWR measurements. These experiments are used to characterize and validate the different UWB chips used. Here we notice the difference between the varying evaluation methods introduced in Chapter 3.2. Table 5.1 shows the evaluation using three different sets of timestamps, collected at varying distances between the nodes. All the timestamps and other times in table are given in device time units (dtu). These represent the smallest unit of time the DWM1000 chips distinguish and are equal to 15.66 ps. For the comparison in Table 5.1 it does not matter how much distance this represents, as this is a linear relationship, which is why the comparison is done in dtu. Based on the six timestamps the reply and round times are calculated and with those the TOF based on the different evaluation schemes. For the SS-TWR we use either the poll and response timestamps (noted by Node A in the table), or the response and final timestamps (Node B). The TOF calculated using SDS-TWR and ADS-TWR is based on all six timestamps. We observe a significant difference in the TOF based on the evaluation method, as we expect due to the clock drift of the individual nodes. Using (3.3, 3.5, 3.7) we calculate the clock drift $e_a$ and $e_b$ for both of the nodes respectively, as well as the real TOF. As this real TOF is only calculated based on the expected clock drifts using the different evaluation methods, other forms of error like an uncalibrated antenna delay still apply, but are not relevant for this comparison between the different evaluation methods. We notice, that the TOF calculated using the SDS-TWR scheme still has a significant error, as the two reply times of the nodes $D_a$ and $D_b$ are not equal. But the TOF of the ADS-TWR is essentially the same as the real TOF, the time difference only represent a few micrometers,

| | Ex. 1 (ca. 2 m) | Ex. 2 (ca. 20 m) | Ex. 3 (ca. 20 m) |
|---|---|---|---|
| $T_{p,tx}$ [dtu] | 649801818676 | 943532611636 | 475785643060 |
| $T_{p,rx}$ [dtu] | 201866349002 | 711864963760 | 383905949084 |
| $T_{r,tx}$ [dtu] | 202129350196 | 712128126004 | 384169093172 |
| $T_{r,rx}$ [dtu] | 650064820785 | 943795780378 | 476048792718 |
| $T_{f,tx}$ [dtu] | 650355932725 | 944086817845 | 476339763765 |
| $T_{f,rx}$ [dtu] | 202420463667 | 712419172433 | 384460073765 |
| $R_a = T_{r,rx} - T_{p,tx}$ [dtu] | 263002109 | 263168742 | 263149658 |
| $D_b = T_{r,tx} - T_{p,rx}$ [dtu] | 263001194 | 263162244 | 263144088 |
| $R_b = T_{f,rx} - T_{r,tx}$ [dtu] | 291113471 | 291046429 | 290980593 |
| $D_a = T_{f,tx} - T_{r,rx}$ [dtu] | 291111940 | 291037467 | 290971047 |
| $T_f$ (SS-TWR Node A (3.2)) [dtu] | 457.5 | 3249.0 | 2785.0 |
| $T_f$ (SS-TWR Node B (3.2)) [dtu] | 765.5 | 4481.0 | 4773.0 |
| $T_f$ (SDS-TWR (3.4)) [dtu] | 611.5 | 3865.0 | 3779.0 |
| $T_f$ (ADS-TWR (3.6)) [dtu] | 603.687349 | 3834.015399 | 3729.079696 |
| $e_a$ [ppm] | −0.669 | −2.47 | −4.36 |
| $e_b$ [ppm] | 0.443 | 1.98 | 2.81 |
| $T_f$ (real) [dtu] | 603.687417 | 3834.016336 | 3729.082581 |

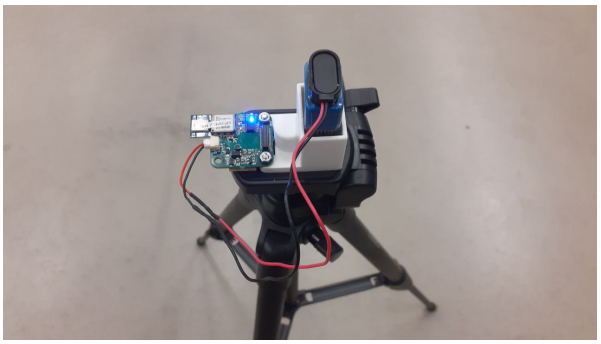
**Table 5.1:** Comparison of the evaluation of the different TWR schemes, based on three TWR measurements.

meaning as expected the error due to clock drift is overcome. The calculated clock drifts in the three test runs vary, as they are done using different UWB boards at different times, but they all fall well below the stated maximum clock drift of ±20 ppm as per [43].

After the validation of the clock drift and the general implementation of the ADS-TWR process, we now preform continuous TWR distance measurements at varying fixed known distance. From now on in all further experiments we always use the ADS-TWR scheme to determine the distances. In a first test two UWB PCBs are lying on the ground inside the Robotics Hall of the University Würzburg next to a tape measure. Figure 5.1a shows the setup of the UWB node at the zero point, the other node is set up in the exact same way a set distance away. For the experiment we take 1000 distance measurements at each of the varying distances, starting at 1 m, incrementing by 1 m each time and ending at 9 m. Figure 5.2 plots the error of the measurements (measured distance − real distance) for the different real distances. For distances of 1 m to 3 m we observe a small spread in the measured values, indicated by a small standard deviation of 1.6 cm, 1.8 cm and 2.6 cm respectively. The error of these measurements is roughly the same at 50 cm, meaning the measured distance is too long. For the larger distances the measurements are less precise with standard deviations larger than 30 cm and also larger errors. We have an explanation for this effect, when we compare this result to the results of the next experiments.



**(a)** Setup of the UWB node at the zero point for the first experiment. The front of the antenna is aligned with the zero mark of the tape measure, with the other node set up the same way at the measured distance.



**(b)** The UWB PCB mounted onto a tripod using a custom 3D printed mount, which also has a slot for the 9 V block battery.

**Figure 5.1:** The setup of the UWB nodes, that are not mounted to the robot.

Just like before we once again measure distances between two UWB nodes at varying fixed known distances. This time the UWB nodes instead of lying on the ground are mounted on a tripod 66 cm above the ground (cf. Figure 5.1b). Otherwise, the setup of the experiment is identical to the one before. This experiment run is repeated twice, once inside the Robotics Hall, where we test up to a distance of 17 m, and once outside up to a distance of 20 m. The resulting errors of the measurements are depicted in Figure 5.3. In these experiments we notice the same behavior as we did for the distances of 1 m to 3 m, when the UWB nodes were lying on the ground. But now this behavior is roughly the same for all distances. The measurements are once again more precise, the standard deviation is at most 4.4 cm, and the error is again fairly constant at 50 cm, especially for the measurements, that are taken inside (cf. Figure 5.3a). The constant offset of the measurements is expected, as the antenna delay of the nodes is not yet calibrated, which leads to such a constant offset (cf. (3.10)). We assume that the inaccurate and imprecise measurements at distances of more than 3 m, when the UWB nodes are lying on the ground (cf. Figure 5.2), stem from increased interference of the ground. Therefore, we use the tripods in all further experiments for all UWB nodes, that are not fixed to the robot.

The final set of experiments done to test the functional principle of the UWB nodes, measures the distances between multiple nodes, not just two. In this experiment we measure the distance between each pair of active nodes to create a full mesh of measurements for the entire UWB network. Using this information we reconstruct a 2D map with the positions of all used UWB nodes. This experiment ensures, that the full cycle of TWR process functions as expected and the UWB nodes report the distance measurements correctly, even if they are not directly connected to the PC, that preforms the evaluation. In each of the experiment runs we take the average of 50 distance measurements for each of the UWB node pairs. As we saw in the previous experiment (cf. Figure 5.3a) the standard deviation of the distance measurements is fairly low, so 50 measurements is enough, to obtain an accurate reading, ignoring the error due to antenna delay. The position and orientation of the nodes in space is arbitrary, which is why we set the position of Node 0 to $(0, 0)$ and the $y$ position of Node 1 (in Map 3 Node 2 instead) to 0. This only fixes the position and orientation of the nodes in space, but has no effect on

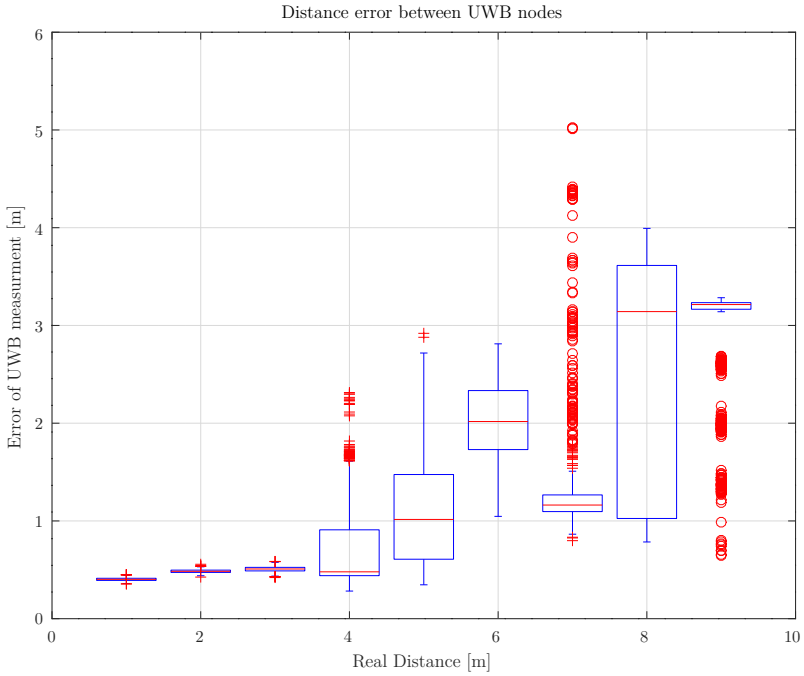
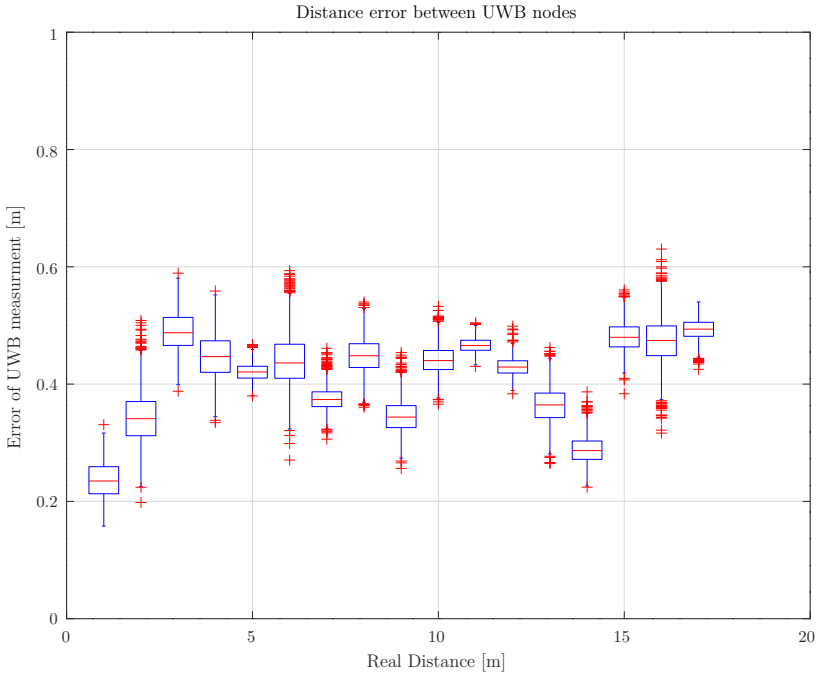
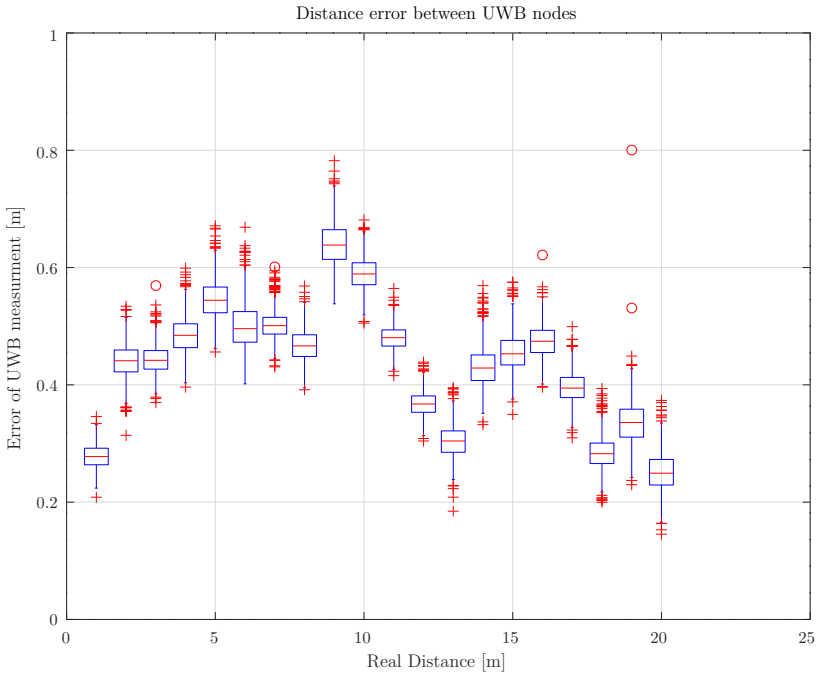



**Figure 5.2:** The error of the measured distances using two UWB nodes lying on the ground at the varying set distances.

the relative position of the nodes. Using these positional constraints we preform a method of least squares to determine the position of all the remaining nodes. As an error function for the least squares method used, we use the sum of the difference of the distance measurement and calculated distance, based on the optimized positions of the nodes, for each pair of nodes. This provides us with a position for all nodes, which is plotted against the real position of the UWB nodes in Figure 5.4. In the first four experiment runs we use a total of four nodes, positioned in a square with a side length of 5 m, in the last two runs a fifth node is added in the center of the square. In the third run the square is rotated by 45 °. The reconstructed position of Node 0 is always exactly at $(0, 0)$, as this is one of the constraints. Therefore, the error Node 0 is always 0, which is why it is left out in Table 5.2. Table 5.2 lists the positional error for each the nodes in each experiment run (map), and the mean error for each map. Especially in Maps 1-4 we notice a significantly smaller error for Node 1, which is expected, as this is the second node with a positional constraint. In Maps 5 and 6 the error for Node 2 is significantly larger. We assume, that the distance measurements, which have another UWB node in their Line of Sight (LOS), are particularly bad. In the fifth and sixth experiment run the additional node in the center of the square is in the LOS for both diagonals. As the position of Node 0 is fixed, the diagonally opposite node (Node 2) is pushed outwards more significantly, thus the larger error value. Overall the mean error for Maps 5 and 6 is larger than for the other maps, which support the assumption, that other UWB nodes in the LOS between two nodes, that preform a distance measurement, worsen that measurement. We also notice, that the mean

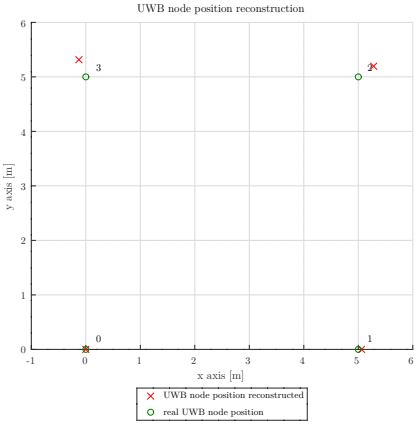



**(a)** Inside the Robotic Hall

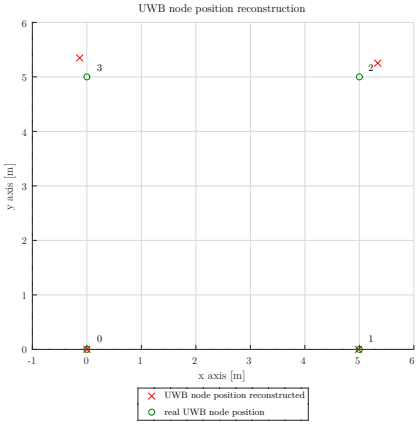



**(b)** Outside

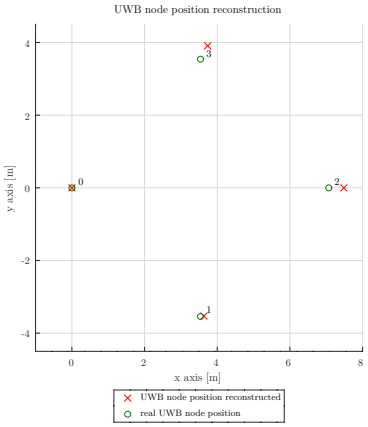
**Figure 5.3:** The error of the measured distances using two UWB nodes mounted on a tripod at the varying set distances.



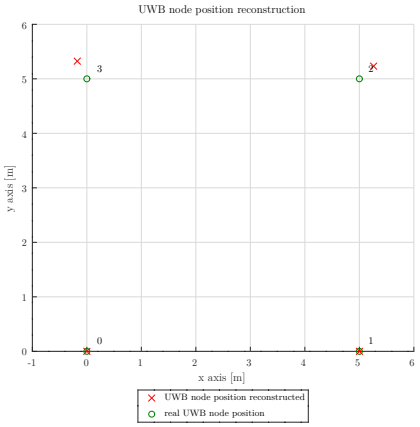

**(a)** Map 1



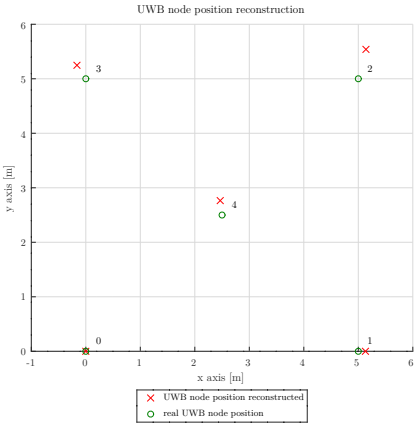

**(b)** Map 2



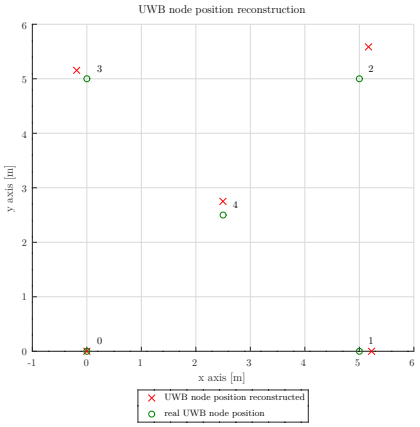

**(c)** Map 3




**(d)** Map 4




**(e)** Map 5




**(f)** Map 6

**Figure 5.4:** The maps showing the reconstructed position of the UWB nodes compared to the real position. The reconstruction is based on a full mesh of distance measurements between each of the nodes.

| | Map 1 | Map 2 | Map 3 | Map 4 | Map 5 | Map 6 |
|---|---|---|---|---|---|---|
| Node 1 | 0.05955 | 0.01640 | 0.09297 | 0.00251 | 0.13106 | 0.22381 |
| Node 2 | 0.34310 | 0.42165 | 0.41339 | 0.34924 | 0.55713 | 0.60827 |
| Node 3 | 0.33950 | 0.37092 | 0.41781 | 0.36860 | 0.29613 | 0.24285 |
| Node 4 | - | - | - | - | 0.26677 | 0.25055 |
| mean | 0.24738 | 0.26966 | 0.30806 | 0.24012 | 0.31277 | 0.33137 |

**Table 5.2:** The position error for the reconstructed UWB nodes for the six maps created (cf. Figure 5.4). Particularly striking values are highlighted.

positional error is smaller than the distance error in the previous experiment, with just two nodes (cf. Figure 5.3a). So a full mesh of measurements pulls the positions slightly together, which somewhat offsets the until now disregarded effect of the antenna delay. This set of experiment runs validates the automatic cycling of TWR measurements on the UWB PCBs introduced in Chapter 4.2.2. An ideally perfect reconstruction of the positions of the UWB nodes is less important, especially as the antenna delay is not yet calibrated.

## 5.2 Antenna Delay Calibration

As previously mentioned it is imperative to calibrate the UWB nodes, such that the effect of the antenna delay is compensated. To achieve this, we must determine the antenna delay of the nodes and set this in their respective embedded code. With this information the timestamps are automatically adjusted, according to Chapter 3.2.3, which corrects for the antenna delay. Therefore, the calibration process consists of determining accurate antenna delays for all the individual UWB chips. To do this we have tested multiple different schemes, introduced in multiple papers [72–74] and the official Application Note from DecaWave [75]. All these systems to determine the antenna delay preform TWR measurements between two or three nodes and then compare the result to the real distance between the nodes, which we measure beforehand using a tape measure. Due to the difference between the real distance and the determined distance using TWR the antenna delay is calculated using slightly different approaches in the different papers. As per the User Manual [19] for the channel and Pulse Repetition Frequency (PRF), that we are using, the UWB nodes are supposed to be 9.3 m apart from each other during the antenna delay calibration, as this minimizes other less relevant error sources. Therefore, we set the nodes up in an equilateral triangle with a side length of 9.3 m, for all antenna delay calibration measurements, that need three nodes, and simply 9.3 m apart from each other, if only two nodes are necessary. The calibration process outlined in the Application Note [75] works iteratively. It calculates the TOF between three nodes and compares that to the expected TOF based on the real distance between the nodes. Using this it adapts the antenna delay, iteratively until a satisfactory outcome is achieved. In [72] a calibration method is introduced, that is based on three UWB messages, that are sent between three nodes. Using the timestamps and the expected TOF based on the real distances, they calculate the antenna delay of the involved nodes. This process is repeated, and the various determined antenna delays are averaged for

| | $d_{1,2}$ [m] | $d_{2,1}$ [m] | $d_{1,3}$ [m] | $d_{3,1}$ [m] | $d_{2,3}$ [m] | $d_{3,2}$ [m] |
|---|---|---|---|---|---|---|
| Run 1 | 9.736 | 9.735 | 9.524 | 9.521 | 9.558 | 9.544 |
| Run 2 | 9.772 | 9.768 | 9.567 | 9.564 | 9.441 | 9.445 |

**Table 5.3:** Averaged distance values between three nodes over 1000 measurements respectively. The two runs were taken after each other using the exact same setup.

each node. For this system to achieve a low systematic error it is necessary, that the reply time of a UWB node is consistent over multiple responses. Even though it does not require equal reply times from two different nodes, like in the SDS-TWR process, this requirement is still unrealistic to achieve, as the exact reply time varies from one reply to another. [73] introduces a different evaluation, of the timestamps of three messages, that three nodes exchange between each other. The evaluation takes the clock drift between the different nodes into account and determines the antenna delay of two of the involved nodes. Once again this process is repeated multiple times, the authors suggest 100 iterations, and the results are averaged. Finally, [74] proposes an antenna delay determination process, that also considers clock drift, but using the ADS-TWR process as a basis. They adapt the formula for ADS-TWR (3.6) so that it also accounts for the combined antenna delay between the two nodes, that preform the ADS-TWR process. They then preform multiple measurements and save the respective timestamps. Using those they minimize the sum of the norm of the difference between the real distance between the two nodes and the calculated distance using the adapted ADS-TWR formula for each measurement. They suggest using Particle Swarm Optimization (PSO) to preform this minimization, which determines the optimal combined antenna delay for the pair of UWB nodes. Preforming the same process for each possible pair of UWB nodes, they deduce the individual antenna delays of the nodes.

We have implemented all the different antenna delay calibration schemes presented here, to ideally compare them and determine a consistent and accurate antenna delay for each UWB node. Unfortunately this was unsuccessful, as the experiments even using the same method do not generate reproducible results. Table 5.3 shows the average over 1000 measured distance values for each combination between the three used nodes exemplary. We repeat the exact same experiment using the same nodes in the same positions, but get different results. Even though the difference between Run 1 and Run 2 is fairly small, it leads to drastically different antenna delays no matter which of the calibration methods presented is used, as they all try to achieve perfect calibration using only a limited number of nodes. Therefore, we use a different method to calibrate the antenna delay. In Figure 5.3 we notice, that all measurements overestimate the actual distance by roughly 50 cm. As per (3.10) we must increase the antenna delay by the time corresponding to half the distance error to mitigate the error. Therefore, we increase the antenna delay for all UWB nodes by 53 dtu. This does not calibrate all nodes for their individual antenna delay, but produces distance measurements accurate enough for our use case.

## 5.3 Runtime Optimizations

The first experiments and the determined antenna delay leave us with a validated functioning UWB code basis, to use for our localization system tests. Before we preform and evaluate the

| | |
|---|---|
| fixed transition time | 871 ms |
| variable transition time | 482 ms |

**Table 5.4:** Potential time savings in the average time out of 200 for a full cycle of TWR measurements with three tags and three anchors, due to variable transition times in between states of the UWB nodes.

experiments for the full system, we optimize the runtime of a full cycle of TWR measurements. For this we adjust the two wait times during the TWR process. First the wait time, by which `nextTimeToAction` is increased, i.e. the transition time in between two states of the UWB nodes (cf. Figure 4.4). Second the wait time of the responding nodes during the response process (`respWaitTimePerNode`), i.e. the time between two consecutive response messages.

As seen in both Algorithm 1 and 2 the node, that starts the TWR process, waits twice to transition into the next state by increasing `nextTimeToAction`. In both cases the wait time is $n \times$ `respWaitTimePerNode` $= n \times 5$ ms. In an earlier version we send the final message after all response messages are received, and the next node is activated after a fixed 150 ms, instead of having both transition times depend on the number of active nodes. Besides an increased average runtime for a full cycle of TWR measurements, which Table 5.4 lists, the earlier implementation also skipped the final message if just a single response message is missing. Due to the optimization of the wait time before the node sends out the next message (either final or activate) we reduce the average full cycle time by more than 40 % (cf. Table 5.4) and in the case of a missing response message, we still send the final message to determine the distance of the nodes, that did respond.

We also optimize the base wait time of the individual nodes before they respond to a message. As explained in Chapter 4.2.2 each node waits `nodeIndex` $\times$ `respWaitTimePerNode` before responding, which means the response messages of the varying nodes are `respWaitTimePerNode` apart from each other. To reliably receive the messages this time can not be too small, but the overall cycle time is decreased the smaller this wait time is. Therefore, we try to reduce this time as much as feasible. To determine this we set up an experiment, in which we count how many successful response messages a Node 0 sending out 1000 poll messages receives from six responding nodes respectively. Table 5.5 lists the results dependent on different base wait times. We notice that for a wait time of 1 ms Node 0 only detects the response messages from Node 1 and 4 reliably. We assume this is because the response messages from Node 2 and 3 interfere with the response from Node 1. Therefore, we only receive every third response message. Similarly, with a wait time of 2 ms the response from Node 2 interferes with the response from Node 1, which leads to only every other the response being detected reliably. From a wait time of 3 ms or longer there are no longer any nodes, whose response message is reliably not detected by Node 0, but on average the reliability still increases with a larger wait time. For our experiments we choose a base response wait time of 5 ms instead of the bare minimum of 3 ms, which increases the overall cycle time in favor of higher operational stability, due to fewer missed messages.

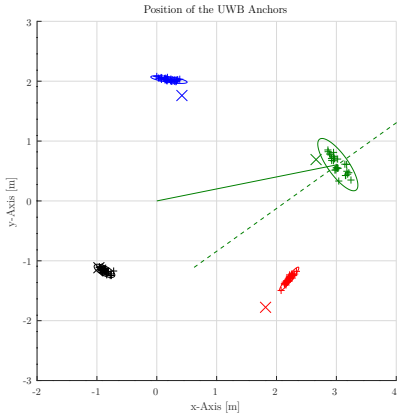
With this all validation, calibration and optimization of the embedded code running on the UWB PCBs is done. We use all the previous experiments to set up the full UWB localization system tests as described in Chapter 4. The embedded UWB system can now also be used to expand upon the evaluation of the distance measurements using the EKF.

| `respWaitTimePerNode` [ms] | successful responses from node # | | | | | | |
|---|---|---|---|---|---|---|---|
| i.e. time in between two response messages | 1 | 2 | 3 | 4 | 5 | 6 | mean |
| 1 | 997 | 2 | 1 | 956 | 32 | 12 | 333.3 |
| 2 | 996 | 2 | 987 | 8 | 987 | 11 | 498.5 |
| 3 | 1000 | 995 | 964 | 784 | 738 | 1000 | 913.5 |
| 4 | 982 | 985 | 922 | 905 | 891 | 990 | 945.8 |
| 5 | 993 | 994 | 978 | 974 | 971 | 994 | 984.0 |
| 10 | 992 | 992 | 969 | 975 | 982 | 999 | 984.8 |

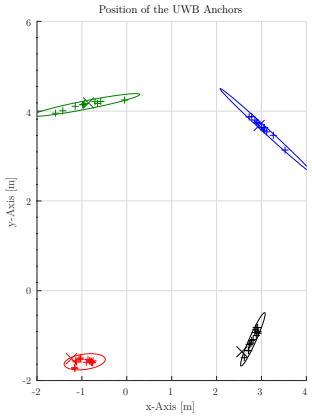
**Table 5.5:** Number of successful responses out of 1000 from each node, depending on the time in between the response messages.

## 5.4 Anchor Position Determination

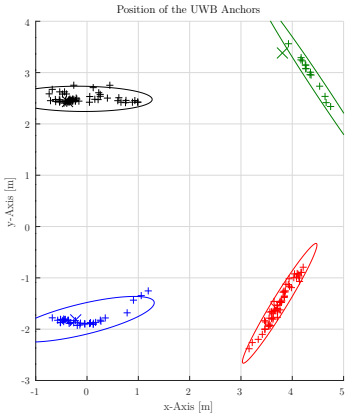
In each test run of the full localization system, described in the following section (cf. Chapter 5.5), we first wait for at least 10 anchor position values before we start to drive the robot. We use the mean of them as the initial estimation for the position of the anchors in the EKF. Here we take a look at the initial position estimation. In Figure 5.5 we show all position estimations for each anchor in the various test runs before the robot starts to move. The robot is at the origin of the coordinate frame for each of the test runs. We also depict the true location of the anchors ($\times$). Using Principal Component Analysis (PCA) we find the principal axis of each data set and determine and plot the corresponding standard deviation along the axis and its perpendicular axis using an ellipse. We list the relevant data in Table 5.6. It lists the number of anchor position measurements before the robot starts to move and the distance error between the mean of the measurements and the true location of the anchor. Also, the standard deviation along the principal axis $\sigma_{principal}$ and its perpendicular axis $\sigma_{\perp}$. Here we notice, that $\sigma_{principal}$ is much larger than $\sigma_{\perp}$, resulting in highly eccentric ellipses in Figure 5.5. We also notice, that the orientation of the principal axis is mostly perpendicular to the direction to the robot, at the origin of the coordinate frame. This is quantified by the angle $\measuredangle O\,(\overline{x}, \overline{y})\ \perp$ in Table 5.6, which is measured between the perpendicular axis (dashed line in Figure 5.5a) and the vector from the origin to the mean of the data set (solid line). As in most cases this is close to $0\,^{\circ}$, this also highlights the distribution of the anchor position measurements. Together this means, that the data points for the position of the anchors scatter mostly along an arc around the robot with constant distance. The setup for the anchor position determination process depicted in Figure 4.5 is the reason for this behavior. As all three tags are compared to the position of the anchors close together and not spread out around them, the three circles representing the distances measurements in Figure 4.5 have a significant overlap and roughly form a large circle around the robot with the tags. Even though there is a clear point on the circle where the minimization problem to find the anchor position (4.1) is optimized, a small change in the measured distances, results in the optimal solution moving roughly along the large circle. This results in the effect we notice in the data sets representing the determined anchor positions in Figure 5.5. The effect is particularly large in the red dataset in Test 6 (cf. Figure 5.5f).



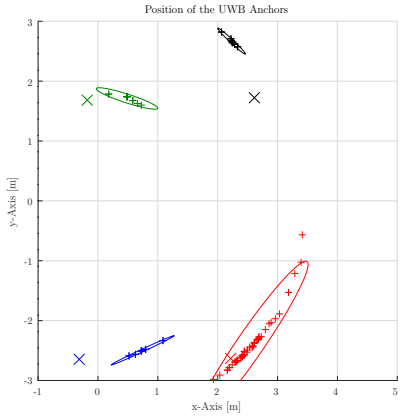

**(a)** Test 1



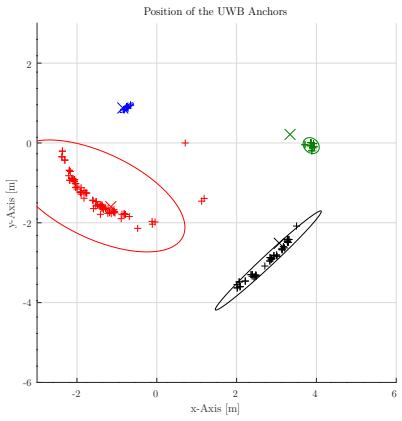

**(b)** Test 2



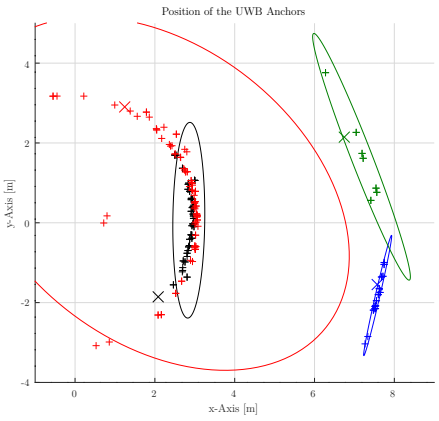

**(c)** Test 3




**(d)** Test 4




**(e)** Test 5




**(f)** Test 6

**Figure 5.5:** The initial determined anchor positions (+) until the robot starts moving. Each color represents a unique anchor, the ellipse the $3\sigma$ range in the principal and its perpendicular axis, positioned at the mean of the measured positions. The true location of the anchor is indicated by a cross (×). In Table 5.6 we list the angle in between the two lines at Anchor 3 in Figure 5.5a for each anchor individually.

| | | | # meas. | $\Delta(\overline{x},\overline{y})$ [cm] | $\sigma_{pricipal}$ [cm] | $\sigma_{\perp}$ [cm] | $\measuredangle O(\overline{x},\overline{y})\perp$ [°] |
|---|---|---|---|---|---|---|---|
| Test 1 | Anchor | 1 (black) | 35 | 10.713598 | 6.350713 | 2.568916 | 9.479874 |
| | | 2 (blue) | 32 | 34.362052 | 10.229677 | 1.584537 | 2.655030 |
| | | 3 (green) | 27 | 37.328925 | 17.428060 | 5.755985 | 24.222944 |
| | | 4 (red) | 33 | 61.638020 | 8.356064 | 1.503850 | 8.221745 |
| Test 2 | Anchor | 1 (black) | 15 | 36.936504 | 21.659944 | 2.590551 | 2.929154 |
| | | 2 (blue) | 13 | 19.190986 | 46.267052 | 1.987646 | 1.518508 |
| | | 3 (green) | 11 | 7.321167 | 40.669161 | 3.122225 | 1.313728 |
| | | 4 (red) | 16 | 30.162678 | 15.367473 | 5.704697 | 38.172773 |
| Test 3 | Anchor | 1 (black) | 61 | 26.724018 | 46.254568 | 8.197769 | 3.204181 |
| | | 2 (blue) | 36 | 13.847013 | 47.679846 | 10.002109 | 16.149916 |
| | | 3 (green) | 21 | 107.326176 | 74.578775 | 4.119900 | 3.600475 |
| | | 4 (red) | 58 | 11.871056 | 45.589046 | 4.709330 | 9.867051 |
| Test 4 | Anchor | 1 (black) | 14 | 102.000407 | 10.572436 | 0.944436 | 3.253040 |
| | | 2 (blue) | 12 | 107.052228 | 19.381369 | 0.737481 | 8.270931 |
| | | 3 (green) | 10 | 66.534537 | 17.979514 | 2.440540 | 1.998116 |
| | | 4 (red) | 50 | 41.633323 | 55.431154 | 7.422643 | 8.585744 |
| Test 5 | Anchor | 1 (black) | 52 | 51.133142 | 60.427897 | 4.737419 | 0.429701 |
| | | 2 (blue) | 12 | 9.494357 | 6.712838 | 1.535025 | 5.773168 |
| | | 3 (green) | 16 | 59.811670 | 7.349745 | 5.930323 | 48.435910 |
| | | 4 (red) | 84 | 41.785047 | 78.579983 | 35.655542 | 22.345725 |
| Test 6 | Anchor | 1 (black) | 74 | 206.716044 | 81.722879 | 13.324532 | 1.934923 |
| | | 2 (blue) | 28 | 27.383016 | 51.571947 | 1.882035 | 0.534695 |
| | | 3 (green) | 14 | 65.490884 | 110.465731 | 8.751532 | 8.254142 |
| | | 4 (red) | 97 | 232.473133 | 179.650552 | 123.053260 | 32.201719 |

**Table 5.6:** Number of measurements, error of the mean compared to the true location $\Delta(\overline{x},\overline{y})$, standard deviation along the principal axis $\sigma_{pricipal}$ and its perpendicular axis $\sigma_{\perp}$ and the angle $\measuredangle O(\overline{x},\overline{y})\perp$ in between the perpendicular axis and the line from the origin to the mean for each anchor in each test run. As an example Figure 5.5a at Anchor 3 shows the two lines, that inscribe the angle $\measuredangle O(\overline{x},\overline{y})\perp$. The color associated with each anchor is the respective color in Figure 5.5.

## 5.5 Full Localization System

Here we describe and evaluate the test runs preformed to validate the full UWB localization system. The general setup is the same for all test runs, but the execution, namely the driven path and whether all anchors are active, varies. Each test run starts with the robot at an arbitrary position, which is defined to be the zero point of the global coordinate system. Then we preform a scan with the Riegl laser scanner for the later scans to compare to. At this point the final system launches the rockets to distribute the UWB sensors, but in our experiments we distribute four anchors at random positions, mounted on the tripods as in the earlier experiments (cf. Figure 5.1b). We now launch the ROS nodes for the anchor position determination and the EKF and after the initialization period of the EKF is done we start to drive the robot. The path, that we drive, where we stop to preform additional laser scans with the Riegl scanner and when we temporarily or indefinitely shut off the anchors, to simulate them failing, is different in each test run. Over the course of the full experiment we let a rosbag record all ROS topics, which enables us to replay each test run multiple times, with a variation of system parameters. The remaining parameters of the EKF, that we set for each test run are as follows: The ROS node publishing the wheel odometry also publishes values for $K_r$ and $K_l$, which are both 0.01 [55]. The EKF subscribes to the topic and uses them in the $\boldsymbol{Q}$ matrix. $\sigma^2_{anPos}$ and $\sigma^2_{dis}$, which are used in the $\boldsymbol{R}$ matrix of the EKF, are also constant over the course of a test run, but we adapt these values to change and optimize the behavior of the EKF over multiple replays. To accomplish this we simply replay the rosbag with only the topics that publish the sensor data (wheel odometry, front 2D laser scanner and the measured UWB distances), so the evaluation is repeated with a new set of parameters. We set the repetition rate of the EKF to 10 Hz, and the number of gathered position values for each anchor, to determine the initial estimation of their position, to 10. Both of these parameters can be changed in the source code, but we keep them the same over the course of all test runs.

In the following we describe each of the individual test runs and discuss key learnings from the individual runs. In Table 5.7 we numerically summarize the performance of different sets of parameters for each test run using a Root Mean Square Error (RMSE) between the trajectory from the EKF and that of the Hector-SLAM algorithm as a ground truth. Figure 5.6 - Figure 5.11 display the trajectories for varying parameters and of different localization strategies for each test run.

**Test 1** In the first test run the robot made a simple roughly 270 ° right-hand loop. This short run is simply to verify the individual parts of the system are working together as intended, with no significant difference between the trajectories expected. The anchors are positioned in roughly a square with side length of 3 m, with the driven path fully within the area between them. As this is the first system test, the anchor to anchor distance measuring feature was not yet implemented, thus in the analysis the variation of $\sigma^2_{dis}$ is skipped. Besides the Riegl scan made at the beginning of the test run, like in all experiments, we only preformed one other scan at the end of the run. Therefore, this test run only consists of a single section. Figure 5.6a shows the trajectories for varying $\sigma^2_{anPos}$, which only has a small effect on the overall shape of the trajectory. This is numerically confirmed by the similar RMSEs in Table 5.7. Even though the overall shape remains similar, locally the trajectory for $\sigma^2_{anPos} = 0.0001$, the smallest of the

evaluated values for this parameter, is far less smooth. This is understandable, as due to the low value for $\sigma^2_{anPos}$ the EKF puts more emphasis on the UWB measurements, which scatter considerably. This phenomenon is also seen in Figure 5.6c, where the trajectory of only using the UWB measurements indeed follows the overall trajectory, but scatters around it significantly. Also, the wheel odometry preforms well in this short test run with a RMSE of 0.19 m, as the expected drift only increases and becomes significant over time. The final position of the Hector-SLAM algorithm is 0.096 m away from the location determined by aligning the two Riegl laser scans, indicating it works well as a continuous ground truth. The anchor RMSE, which is only evaluated at the last frame of any test run section is 0.34 m. This is higher than the robots RMSE, meaning the position estimation of the EKF for the individual anchors is less precise, than that for the resulting trajectory.

In the description and evaluation of the following test runs we will notice many effects, like the non-smooth trajectory for low $\sigma^2_{anPos}$, again. We will not make a particular note of these effects in the following and instead focus on the unique learnings from each test run.

**Test 2** From the second test run onward we integrated the anchor to anchor distance measurements, so these are now also part of the analysis. In this run we drove a rough 'S' shape, with first a left- and then a right-hand turn. Once again the run consists of only a single section 75.9 s long. In this run we notice the expected drift in the wheel odometry of the robot, with its RMSE at 1.08 m (cf. Table 5.7). We also notice a clear difference in the trajectories for different $\sigma^2_{anPos}$ values, shown in Figure 5.7a. The trajectory for $\sigma^2_{anPos} = 100$, is much closer at the wheel odometry, as the EKF puts a greater emphasis on its input. Besides the somewhat expected non-smoothness of the trajectory for $\sigma^2_{anPos} = 0.01$ it also has a kink towards the end of the trajectory. We currently do not have an explanation for it, but it does speak to the overall increased risk and unreliability, if $\sigma^2_{anPos}$ is lowered too far, thus putting too much emphasis on the UWB measurements. For the evaluation of the trajectories with varying values for $\sigma^2_{dis}$, $\sigma^2_{anPos}$ is set to 1, as this is a good compromise between the two extremes. Figure 5.7b shows the trajectories for varying $\sigma^2_{dis}$ values and the one with the inter-anchor distance measurements disabled, to have a comparison. We see, that the effect of the inter-anchor distance measurements is only small, but positive. The best preforming value is $\sigma^2_{dis} = 0.005$ with a RMSE of 0.36 m, compared to 0.51 m with the inter-anchor distance measurements disabled. We also observe the stark similarity between the different trajectories in Figure 5.7b. Figure 5.7d shows the trajectory for $\sigma^2_{anPos} = 1, \sigma^2_{dis} = 0.005$ with the error at each point along the trajectory represented by the color. In it, we notice, that the error does also increase over time, showing that it also is affected by some overall drift. We also see, that the covariance of the EKF, represented by the ellipses in Figure 5.7d, remains at an equal magnitude over the entire test run.

**Test 3** With 107.9 s long the third test run is the longest, of those with only a single section. It also features the most complicated driven path, with two consecutive left-handed loops (cf. Figure 5.8c). Different to the tests before the trajectories for small $\sigma^2_{anPos}$ values, though again less smooth, have a lower RMSE, than that of $\sigma^2_{anPos} = 1$, 0.38 m and 0.31 m compared to 0.43 m (cf. Table 5.7). We also see, that this time the inclusion of the inter-anchor distance measurements has essentially no effect, as the RMSE stays the exact same. Overall the third test run behaves as expected with the same phenomenons as in the previous tests. Also, the error of

the EKF-SLAM trajectory at the end of the test run (cf. Figure 5.8d) is in the same magnitude as in the last test run. The largest difference is the larger RMSE of the wheel odometry at 1.59 m up from 1.08 m, which is as expected due to the longer and more complicated driven path.

**Test 4** Test 4 is the first test run, which consists of three instead of just one section. At two points during the test run we stop and take an additional laser scan with the Riegl VZ-400. In the first section we drive along a right-hand loop before finishing with a sharp left turn. During the first intermediate scan we also disconnect all the UWB anchors from their power supply, simulating them failing. Therefore, the system relies entirely on the wheel odometry over the course of the second test segment. In this we drive mainly straight with another left turn in roughly the middle. After the segment we take the second intermediate laser scan and reconnect the UWB anchors to power. In the third segment UWB measurements are once again available to the EKF. We path we drive follows a wide roughly 180 ° left turn. With the temporary shutoff of the UWB nodes we analyze how the localization system copes with such an outage. Figure 5.9a and Figure 5.9b show the trajectories for the tested values for $\sigma^2_{anPos}$ and $\sigma^2_{dis}$. With these we notice the expected behavior, just like in the previous tests, with higher $\sigma^2_{anPos}$ values tending more towards the trajectory of the wheel odometry, and lower $\sigma^2_{anPos}$ values being less smooth. With Figure 5.9c and Figure 5.9d we analyze the outage of the UWB measurements. During the first section from the starting point (the origin of the coordinate frame) to the first scan location (top left) the trajectories of both the EKF-SLAM and the wheel odometry are unique and stay close to the truth, though slightly behind it. The RMSE are correspondingly low, 0.15 m for the wheel odometry (cf. Table 5.7). Over the course of the second section, ending at the second intermediate scan location (on the right side of the figure), the shape of the trajectory of the EKF-SLAM is identical to that of the wheel odometry. This is expected as there are no UWB measurements, so the system relies only on the wheel odometry. Though the shapes of the trajectories are identical, they do not overlap, as the starting point after the first intermediate scan is slightly different. Also, the trajectory drifts away from the truth considerably, indicated by the larger RMSE of both the EKF and the wheel odometry (cf. Table 5.7). From Figure 5.9d, in which the ellipses represent the variance of the EKF, we notice, that throughout the first segment the variance stays roughly the same, just like in the previous test runs. But during the second section the covariance increases. As the correction step in the EKF is not taking place in this section, as there are no new UWB measurements, the covariance increases in each iteration based on the variance of the wheel odometry represented by $K_r$ and $K_l$. After the second scan in the third section UWB measurements are once again available, so the trajectory of the EKF is unique again. Immediately after the scan the position of the robot in the EKF shifts back by about 1 m towards the true location. This is clearly observed in the change of color of the trajectory in Figure 5.9d. Though this instantaneous correction is not large enough the trajectory gets a little closer to the truth during the third section. This is also indicated by the overall lower RMSE of the third section compared to the second (cf. Table 5.7). The immediate correction is much larger in the case that $\sigma^2_{anPos}$ is 0.0001 or 0.01, as this lets the EKF put far more emphasis on the UWB values (cf. Figure 5.9a). Numerically this is indicated by the lower RMSE for these trajectories (0.54 m and 0.58 m), compared to the trajectories for $\sigma^2_{anPos} = 1$ or 100 (1.07 m and 1.45 m)(cf. Table 5.7). Also, the covariance of

the EKF get smaller as soon as the second intermediate scan is done, but it stays larger than in the first section. After the decrease immediately after the scan it stays the same size, indicating a new equilibrium of the EKF is achieved, just like during the first section as well as during the previous test runs with only one section. The wheel odometry further drifts away from the truth further increasing. The RMSE of the anchors stay mainly the same over the entire test run, indicating the position of the anchors is not really changed by the EKF over the test run. This means it mainly is given by the location they have after the initialization period. This is confirmed by comparing the anchor position in Figure 5.9d to the average anchor position in Figure 5.5d.

Overall Test 4 shows, that during an outage of the UWB anchors the EKF remains functional, but increases its covariance. After an outage of the system is over, the EKF successfully reintegrates the UWB measurements and is able to somewhat correct for the accumulated drift error of the wheel odometry, that is the only input into the EKF, during the UWB outage.

**Test 5** The setup for the fifth test is the same as in the fourth. We once again take two intermediate laser scans with the Riegl VZ-400, at which point we first disconnect the UWB anchors from the power and then reconnect them again during the second scan. This means, that during the second segment of this test run the EKF is entirely relying on the wheel odometry. So we expect to see similar results as in Test 4. The path we drive in this test starts with roughly 270 ° left-hand loop followed by a long straight bit. In the middle of the straight part we take the first intermediate scan, thus starting section two. Here we preform a 180 ° left turn, which is followed by a long final fairly straight final stretch. Along this stretch we preform the second intermediate laser scan. Once again Figure 5.10a and Figure 5.10b show the trajectories for the varying parameters of the EKF. They have the overall shape and smoothness as we expect from the findings in the last test runs, with the correction after the second scan once again being much larger for lower $\sigma^2_{anPos}$ values. In this test run we also notice a clear differentiation between the trajectories for varying $\sigma^2_{dis}$ values, though only in the second section of the run. This is makes sense as the small variations in starting position after the first intermediate scan are amplified during the second section, during which the trajectory has the same shape as that of the wheel odometry. After the second scan though they all recombine to have essentially no impact on the trajectory. The trajectories for lower $\sigma^2_{anPos}$ values once again see a greater correction after the second intermediate scan, though this time the effect on the RMSE of the third section is smaller, than in Test 4 (cf. Table 5.7). Figure 5.10c also highlights the drift of the wheel odometry nicely, as during the second section that trajectory crosses over itself, which the true trajectory does not. Figure 5.9d shows just like in Test 4 the covariance of the EKF rising during the second section, with the largest error at the end of it. After the second scan both the error and covariance immediately reduce, staying fairly consistent over the course of the third section. Overall this test does not provide any new findings, but supports those of the previous test run.

**Test 6** Test 6 is the last test run of the full localization system, which also follows the three sectioned approach with a temporarily disabled UWB measurements during Section 2 as the two previous test runs. It is the longest of all test runs at 264.0 s, and covers the greatest distance. The anchors are also spread out over the largest area, roughly a square with 5 m side length.

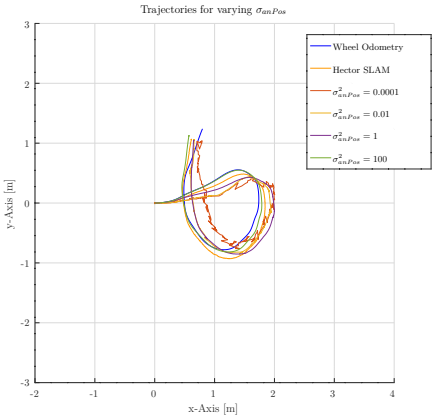
The true trajectory crosses the area in a slightly winding line in the first section, does a left hook to turn around in the second section and finally a wide left turn in the third section. Notable in this run is the bad anchor position estimation, which all ready starts with the initial position estimation (cf. Figure 5.5f). This is never properly corrected by the EKF over the course of the run, thus the RMSE of the anchors (5.83 m) is the highest out of all test runs. This has a large negative effect on the trajectory for the lower $\sigma^2_{anPos}$ values which preform much worse than higher $\sigma^2_{anPos}$ values, even though usually and especially in the third section of Tests 4 and 5, they have lower RMSEs (cf.Table 5.7). The remaining trajectories (Figure 5.11b - Figure 5.11d) show the same phenomenons as in the previous test runs. Although the trajectories for varying $\sigma^2_{dis}$ values remain less bundled as in previous test. Overall they seem to have a slight positive impact during Section 1 and 2, but a negative one in Section 3 (cf. Table 5.7). We do not have an explanation for this change in their effectiveness during the run, but the overall effect remains small.

Over the course of all test runs we showed the feasibility of the UWB localization system. We have noted many phenomenons, that repeated itself in the different test runs. Overall we notice, that the smaller $\sigma^2_{anPos}$ values preformed better, in particular in correcting the EKF estimation after an outage. But their trajectories are less smooth, which may be a problem in some scenarios as further systems, that use the pose data of the robot, generally do not react well to sudden changes in the incoming data. Also, their trajectories are more prone to large errors, if the anchor position determination process works particularly poorly. For the analysis of the inter-anchor distance measurements we settle on $\sigma^2_{anPos} = 1$, as this is a lower value, but not so low, that it fails to work in situations with large errors in the anchor position determination process. This represents a standard deviation of 1 m for the anchor position measurements, which is reasonable looking at the initial estimations of the anchor positions in Figure 5.5. The inclusion of the inter-anchor distance measurements to the system does not have a large effect on the result. It some cases it makes the system preform slightly better in some slightly worse. In the further analysis we use $\sigma^2_{dis} = 0.005$ which represents a standard deviation in the distance measurements of just over 7 cm, which is in line with the maximum standard deviations in our distance measurement tests (cf. Chapter 5.1). The system is also able to cope with outages of UWB nodes, though it does not recover to the same standard as before the outage. We highlight this in Figure 5.12, which shows the final trajectories from the EKF-SLAM for $\sigma^2_{anPos} = 1$ and $\sigma^2_{dis} = 0.005$, with the color representing the error on the same scale. This clearly shows the difference between Test 1 through 3 without an outage and Test 4 to 6. So far we have not mentioned any directionality of the error. Closely looking at the position of the robot according to the EKF at each Riegl laser scan (shown with a circle (∘) in Figure 5.6c - Figure 5.11c) we notice that the EKF usually is behind the true position in the direction of travel. This does not apply to the positions after the second intermediate scan in Tests 4 to 6, as in those cases the EKF only has the wheel odometry as an input during the previous section. Therefore, we infer, that the UWB measurements are responsible for the EKF staying behind the true position. Table 5.4 lists the time for a full cycle of measurements as 482 ms for a system with three tags and three anchors. Considering we need 56 instead of 42 messages for a full cycle of measurements when using seven total nodes instead of six, we assume the minimal time for a full cycle at 643 ms. This means the EKF only receives the anchor position measurements almost a

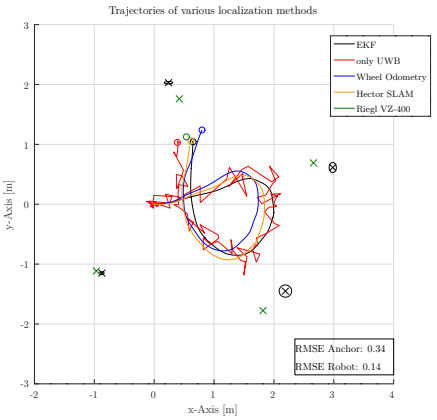
second after they were taken. Therefore, it makes sense, that the EKF position estimation stays slightly behind the true position of the robot.

| Test # - Section # | Run Time [s] | RMSE Wheel Odometry [m] | RMSE EKF [m], without inter-anchor measurements, varying $\sigma^2_{anPos}$ | | | | RMSE EKF[m], $\sigma^2_{anPos} = 1$, varying $\sigma^2_{dis}$ | | | | Error Hector-SLAM [m] | RMSE Anchor [m] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0.0001 | 0.01 | 1 | 100 | 0.001 | 0.005 | 0.05 | 0.5 | | |
| 1 | 65.5 | 0.19 | 0.26 | 0.14 | 0.14 | 0.09 | - | - | - | - | 0.096 | 0.34 |
| 2 | 75.9 | 1.08 | - | 0.57 | 0.51 | 0.95 | 0.44 | 0.36 | 0.44 | 0.49 | 0.078 | 0.37 |
| 3 | 107.9 | 1.59 | 0.38 | 0.31 | 0.43 | 0.73 | - | 0.43 | 0.43 | 0.43 | 0.215 | 0.46 |
| 4 - 1 | 61.2 | 0.15 | 0.42 | 0.31 | 0.33 | 0.11 | 0.26 | 0.29 | 0.31 | 0.29 | 0.174 | 3.15 |
| 4 - 2 | 29.8 | 0.89 | 1.06 | 1.17 | 1.30 | 1.11 | 1.26 | 1.30 | 1.25 | 1.24 | 0.042 | 3.15 |
| 4 - 3 | 49.9 | 1.88 | 0.54 | 0.58 | 1.07 | 1.45 | 0.88 | 0.95 | 0.98 | 1.00 | 0.199 | 2.99 |
| 4 - total | 238.8 | 1.20 | 0.65 | 0.67 | 0.90 | 1.01 | 0.80 | 0.85 | 0.84 | 0.85 | - | - |
| 5 - 1 | 44.1 | 0.88 | 0.25 | 0.23 | 0.08 | 0.43 | 0.25 | 0.22 | 0.23 | 0.11 | 0.057 | 2.98 |
| 5 - 2 | 48.2 | 3.13 | 0.72 | 1.15 | 0.86 | 1.22 | 1.58 | 1.41 | 1.39 | 1.01 | 0.158 | 2.78 |
| 5 - 3 | 41.4 | 3.78 | 1.18 | 1.14 | 1.28 | 1.32 | 1.39 | 1.43 | 1.41 | 1.33 | 0.079 | 3.10 |
| 5 - total | 223.8 | 2.87 | 0.81 | 0.95 | 0.90 | 1.07 | 1.24 | 1.17 | 1.15 | 0.97 | - | - |
| 6 - 1 | 60.3 | 0.81 | 2.59 | 2.04 | 0.61 | 1.00 | 0.37 | 0.41 | 0.41 | 0.42 | 0.205 | 5.71 |
| 6 - 2 | 62.7 | 2.88 | 4.40 | 4.61 | 1.68 | 2.79 | 1.24 | 1.25 | 1.21 | 1.21 | 0.219 | 5.49 |
| 6 - 3 | 46.5 | 4.42 | 2.06 | 2.49 | 0.91 | 1.24 | 1.45 | 1.83 | 1.48 | 1.34 | 0.216 | 5.83 |
| 6 - total | 264.0 | 2.94 | 3.27 | 3.32 | 1.18 | 1.91 | 1.08 | 1.25 | 1.09 | 1.04 | - | - |

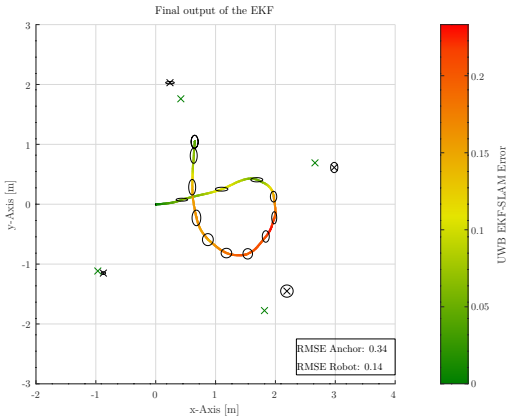
**Table 5.7:** Numerical results of the full localization tests. The RMSE of the wheel Odometry and the EKF is calculated using the trajectory of the Hector-SLAM algorithm as the ground truth. For the calculation only points, where the robot is moving, are included. This excludes all times, when we capture a laser scan with the Riegl VZ-400. The test runs are broken up into individual sections between each of these scans. At each scan the error of the Hector-SLAM algorithm is the distance to the point, where the Riegl scan is taken. For the RMSE of the anchors we use the distances to the ground truth supplied by the Riegl laser scan.




**(a)** Trajectories for varying $\sigma^2_{anPos}$, with no inter-anchor distance measurements.

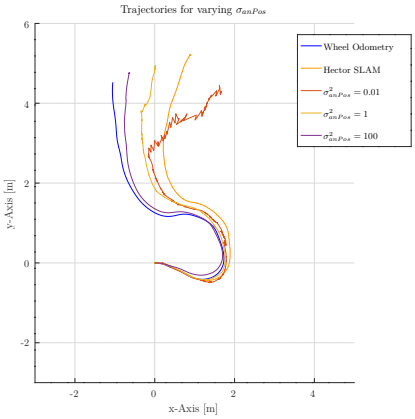



**(c)** Trajectories for various localization methods. The crosses (×) represent the anchors, the circles (○) the position of the various methods, at the time we capture a Riegl scan.

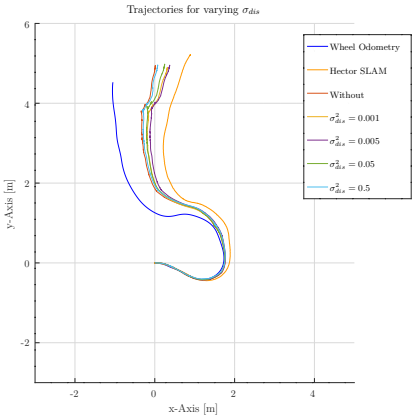



**(d)** The output trajectory of the EKF for $\sigma^2_{anPos} = 1$. The color represents the distance error to the trajectory of the Hector-SLAM algorithm. The ellipses the $3\sigma$ range in the $x$- and $y$-axis, based on the covariance matrix of the EKF.

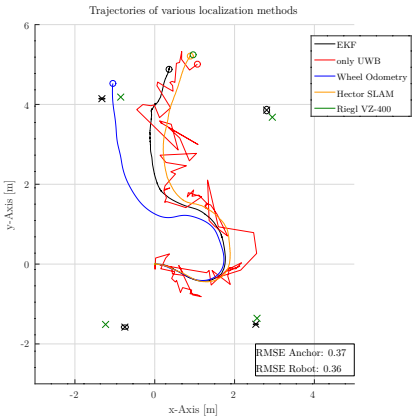
**Figure 5.6:** Localization system Test 1 results.



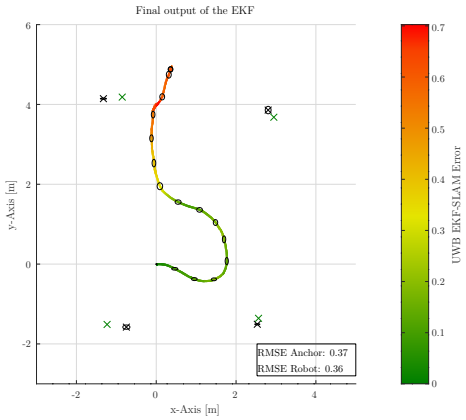

**(a)** Trajectories for varying $\sigma^2_{anPos}$, with no inter-anchor distance measurements.




**(b)** Trajectories for varying $\sigma^2_{dis}$, with $\sigma^2_{anPos} = 1$.

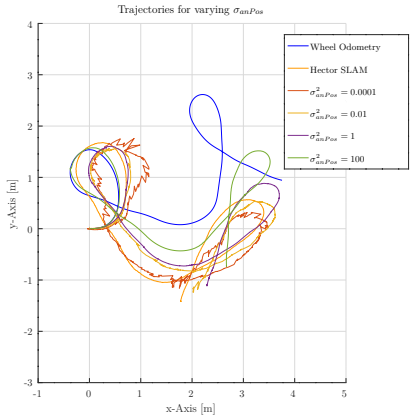



**(c)** Trajectories for various localization methods. The crosses (×) represent the anchors, the circles (○) the position of the various methods, at the time we capture a Riegl scan.

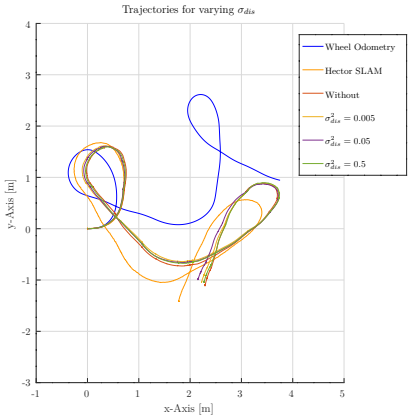



**(d)** The output trajectory of the EKF for $\sigma^2_{anPos} = 1$ and $\sigma^2_{dis} = 0.005$. The color represents the distance error to the trajectory of the Hector-SLAM algorithm. The ellipses the $3\sigma$ range in the $x$- and $y$-axis, based on the covariance matrix of the EKF.

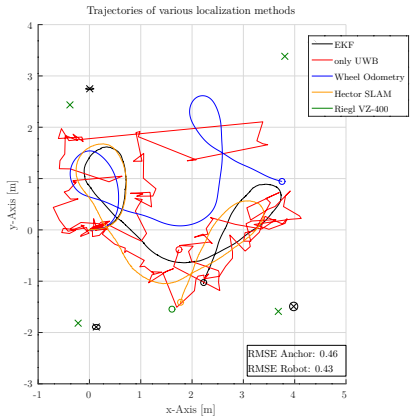
**Figure 5.7:** Localization system Test 2 results.



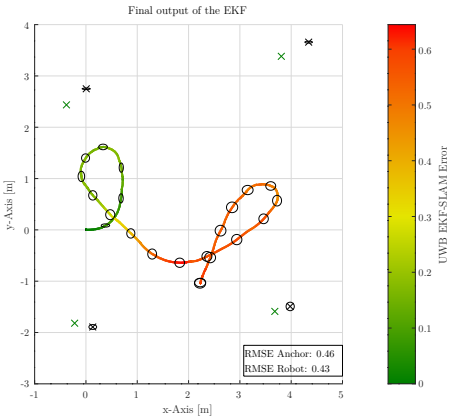

**(a)** Trajectories for varying $\sigma^2_{anPos}$, with no inter-anchor distance measurements.




**(b)** Trajectories for varying $\sigma^2_{dis}$, with $\sigma^2_{anPos} = 1$.

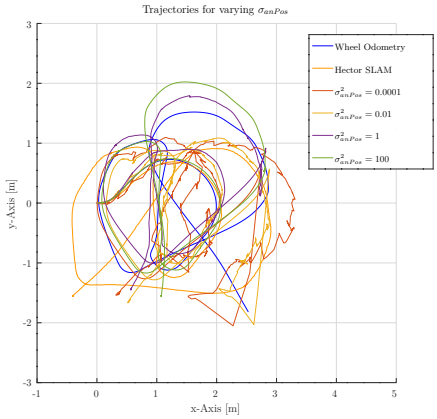



**(c)** Trajectories for various localization methods. The crosses (×) represent the anchors, the circles (○) the position of the various methods, at the time we capture a Riegl scan.

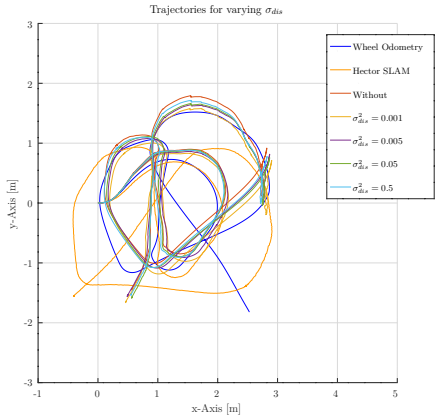



**(d)** The output trajectory of the EKF for $\sigma^2_{anPos} = 1$ and $\sigma^2_{dis} = 0.005$. The color represents the distance error to the trajectory of the Hector-SLAM algorithm. The ellipses the $3\sigma$ range in the $x$- and $y$-axis, based on the covariance matrix of the EKF.

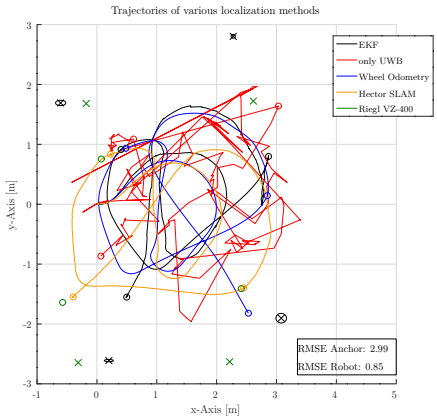
**Figure 5.8:** Localization system Test 3 results.



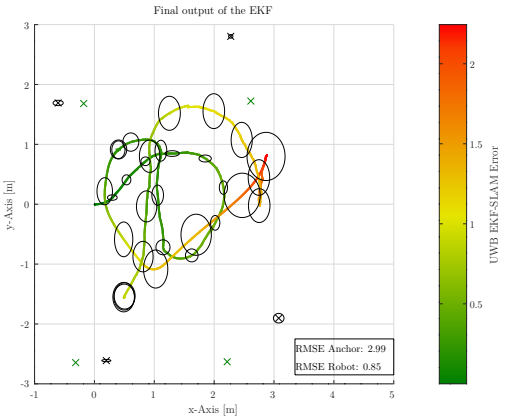

**(a)** Trajectories for varying $\sigma^2_{anPos}$, with no inter-anchor distance measurements.




**(b)** Trajectories for varying $\sigma^2_{dis}$, with $\sigma^2_{anPos} = 1$.

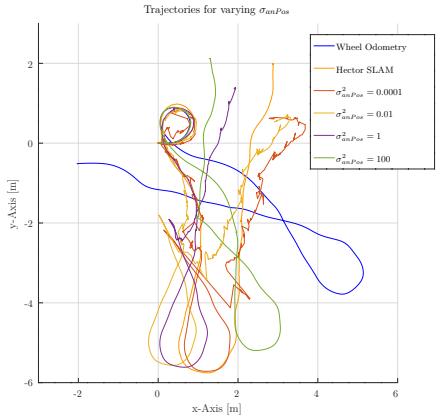



**(c)** Trajectories for various localization methods. The crosses (×) represent the anchors, the circles (○) the position of the various methods, at the times we capture a Riegl scan.

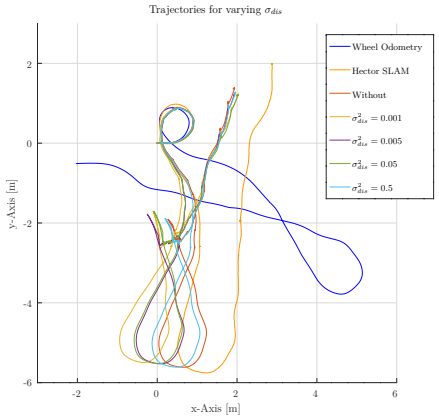



**(d)** The output trajectory of the EKF for $\sigma^2_{anPos} = 1$ and $\sigma^2_{dis} = 0.005$. The color represents the distance error to the trajectory of the Hector-SLAM algorithm. The ellipses the $3\sigma$ range in the $x$- and $y$-axis, based on the covariance matrix of the EKF.

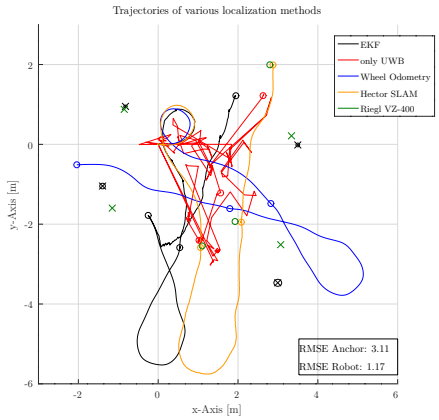
**Figure 5.9:** Localization system Test 4 results.



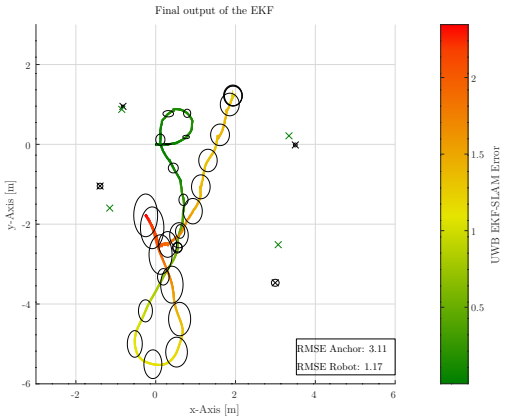

**(a)** Trajectories for varying $\sigma^2_{anPos}$, with no inter-anchor distance measurements.




**(b)** Trajectories for varying $\sigma^2_{dis}$, with $\sigma^2_{anPos} = 1$.

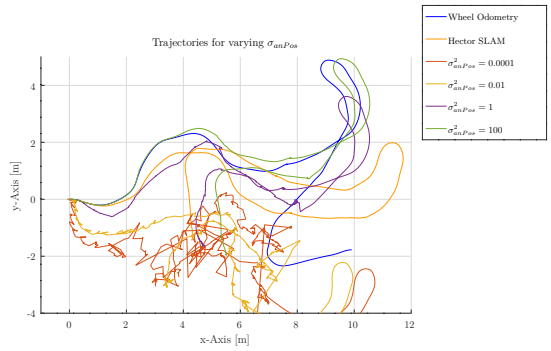



**(c)** Trajectories for various localization methods. The crosses (×) represent the anchors, the circles (○) the position of the various methods, at the times we capture a Riegl scan.

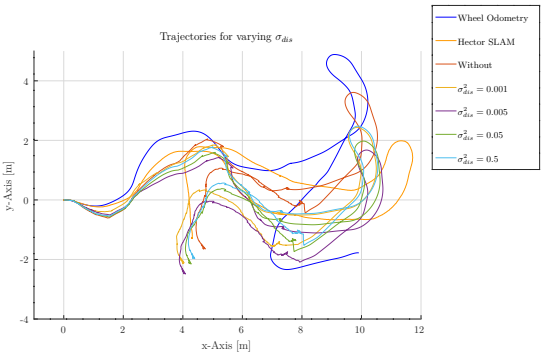



**(d)** The output trajectory of the EKF for $\sigma^2_{anPos} = 1$ and $\sigma^2_{dis} = 0.005$. The color represents the distance error to the trajectory of the Hector-SLAM algorithm. The ellipses the $3\sigma$ range in the $x$- and $y$-axis, based on the covariance matrix of the EKF.

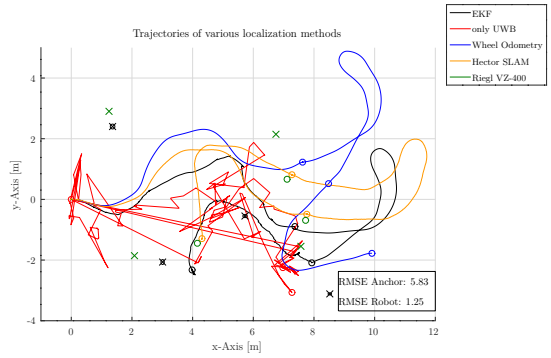
**Figure 5.10:** Localization system Test 5 results.



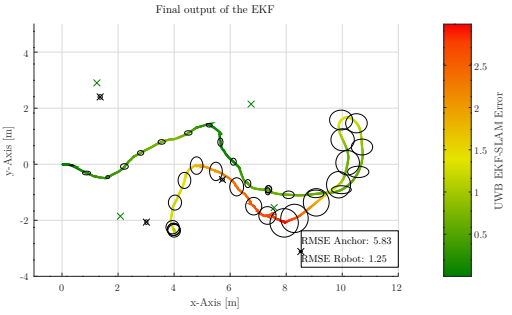

**(a)** Trajectories for varying $\sigma^2_{anPos}$, with no inter-anchor distance measurements.




**(b)** Trajectories for varying $\sigma^2_{dis}$, with $\sigma^2_{anPos} = 1$.

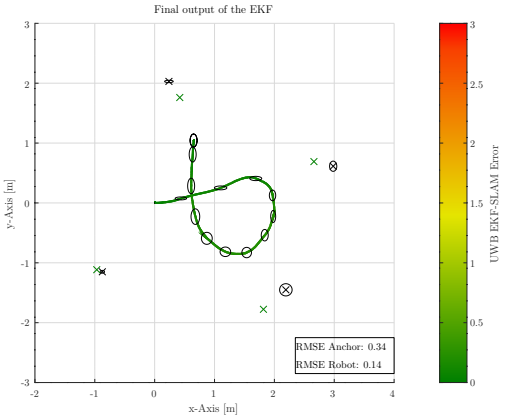



**(c)** Trajectories for various localization methods. The crosses (×) represent the anchors, the circles (○) the position of the various methods, at the times we capture a Riegl scan.

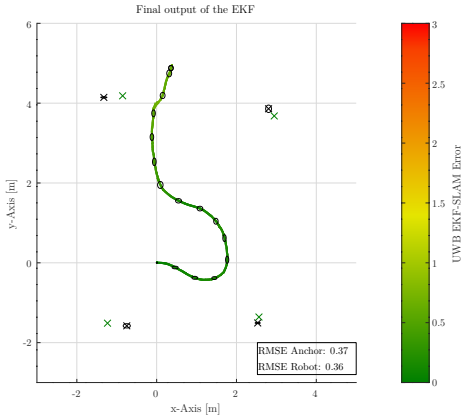



**(d)** The output trajectory of the EKF for $\sigma^2_{anPos} = 1$ and $\sigma^2_{dis} = 0.005$. The color represents the distance error to the trajectory of the Hector-SLAM algorithm. The ellipses the $3\sigma$ range in the $x$- and $y$-axis, based on the covariance matrix of the EKF.

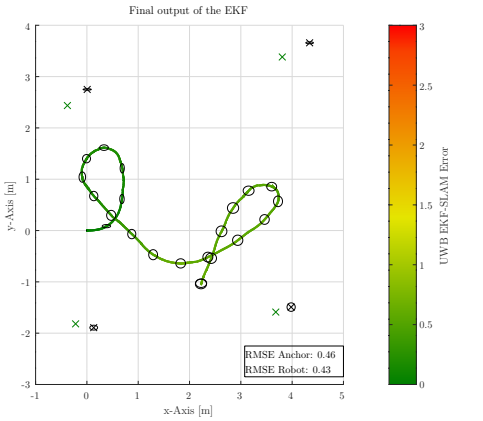
**Figure 5.11:** Localization system Test 6 results.



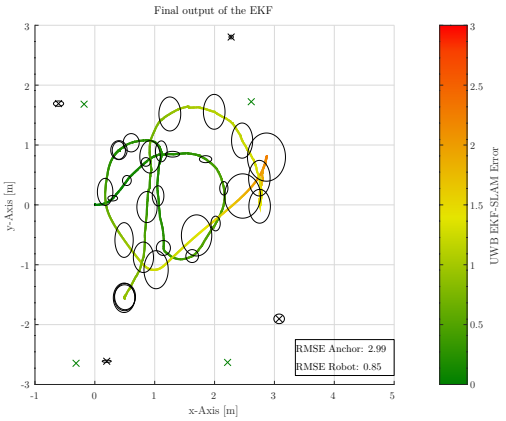

**(a)** Test 1



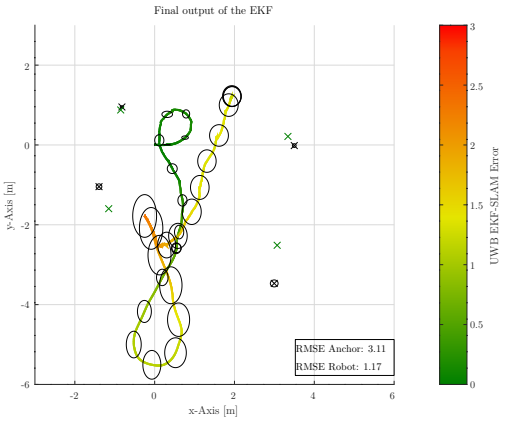

**(b)** Test 2



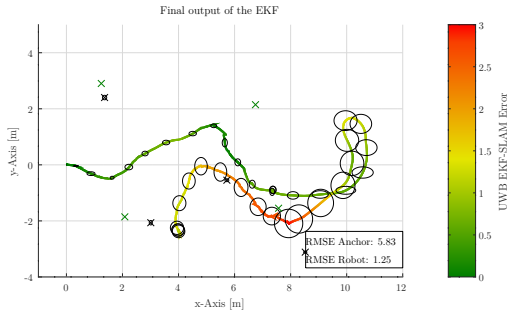

**(c)** Test 3




**(d)** Test 4




**(e)** Test 5




**(f)** Test 6

**Figure 5.12:** The output trajectory of the EKF for $\sigma^2_{anPos} = 1$ and $\sigma^2_{dis} = 0.005$. The color, which in this figure is on the same scale over all test runs, represents the distance error to the trajectory of the Hector-SLAM algorithm. The ellipses the $3\sigma$ range in the $x$- and $y$-axis, based on the covariance matrix of the EKF.

# Chapter 6

# Conclusion

In this thesis we introduce and implement a novel way to combine UWB ranging measurements with an EKF-SLAM algorithm to preform localization of a robot. In currently common UWB localization systems it is required to know the exact position of the anchors a priori, which are then used to locate a moving tag. For this the tag preforms a ranging scheme with the anchors, to use the distances to them and their positions to calculate its position in a trilateration process similar to GNSS. For remote and (semi)-autonomous operations of a robot, like those in planetary exploration, it is not feasible to first set up the anchors at known locations, but rather the robot has to bring its own localization system. To achieve this we set up a system, that not only preforms ranging from the tag to the anchors, but also in between the anchors, and uses a novel evaluation method to first determine the position of the randomly distributed anchors, before using them in an EKF for position estimation. For this we implement a concurrent TWR approach on a custom PCB [59], equipped with a DWM1000 UWB IC [4] and an STM32 microprocessor [60]. We discuss multiple TWR schemes and how they relate to an error due to clock drift and what effect missing antenna delay calibration has on the system. At any time one UWB node preforms a concurrent TWR operation with all other nodes to determine its distances to them. These are then relayed to the PC running the robot, after which the node passes the task along to the next node, which repeats the process. Therefore, we collect distance measurements for all pairs of nodes in the system. We also implement a stability feature, that ensures continuous operations in the case, that individual UWB messages are corrupted or lost and also in the case that a one or more UWB nodes fail completely. Our localization system consists of three UWB nodes mounted in an equilateral triangle on the robot, and multiple anchors spread out in the robots surroundings at random unknown positions. In the fully integrated system 3D printed rockets distribute the UWB anchors, launching from the robot, thus the robot brings its own localization system and sets it up remotely. Using the TWR distance measurements from the three tags, at known positions on the robot, to the anchors, we determine the position of each anchor individually. This produces the relative position of multiple landmarks (namely the UWB anchors) to the robot. We use this as the observation input of an EKF-SLAM algorithm, whose state contains the pose of the robot and positions of the anchors. The system input of the EKF is the wheel odometry of the robot. We also explain the formulation and workings of an EKF with a non-additive noise formulation and the

capability of handling missing observations, which is the theoretical basis for the EKF used in our localization system. For the ground truth we use two systems based on laser scanners. First a continuous 2D laser scanner, whose data is used in the Hector-SLAM algorithm [69]. As all our experiments are done indoors, where there are a lot of features such as walls in the scanning plane, this produces a good continuous estimation of the robots pose. Second we use high definition 3D laser scans from the Riegl VZ-400 [58] at discrete time points, which produces precise pose information of the robot at those time points, and also the position of the UWB anchors.

We preform multiple different experiments, which aim to validate the individual components of the full system, before we run full system tests. In the first experiments we validate the performance of the different TWR evaluation methods, which confirm the varying dependency on clock drift of them. In the case of SS-TWR and SDS-TWR, the resulting TOF has a considerable error, while the error is negligible with ADS-TWR. We also confirm, that the clock drift of the DWM1000 transceivers lie within $\pm 5$ ppm, well below the specified maximum of $\pm 20$ ppm by [43]. During first tests of the distance measuring, we notice the effect of missing antenna calibration, as the distance values are all roughly 50 cm above the true distance. We also notice a larger error and poorer consistency, if the UWB nodes lie on the ground, rather than mounted on tripods. During the antenna calibration process we notice, that even with an exact repeat of experiment conditions the measured distances vary, though only slightly. For all tested antenna calibration methods, which try to perfectly determine the antenna delay of just a limited set of UWB nodes at a time, this leads to unusable results. Therefore, we do not preform node specific antenna calibration, but rather set the antenna delay, such that the 50 cm error is eliminated.

We preform six full system tests, in each of which we set up four anchors on tripods in random positions around the robot. Both at the beginning and the end of a test run we preform a Riegl laser scan, to determine the exact final position of the robot. During the test runs we drive varying paths inside the Robotics Hall at the University of Würzburg, while the UWB localization system is running. In the second set of three tests we make two intermediate laser scans respectively, which divide the test run into three sections. During Section 2 we disconnect the UWB anchors from their respective power source, thus simulating them failing temporarily. We analyze the results for varying values for the parameters $\sigma^2_{anPos}$ and $\sigma^2_{dis}$ of the EKF.

The position of the robot determined by the Hector-SLAM algorithm has a difference of less than 30 cm to that determined by aligning Riegl scans at all scanning locations. This means it is ideally suited as a continuous ground truth in the feature rich indoor testing environment. As expected the wheel odometry preforms poorly with a RMSE as high as 2.94 m, during the longest test run of 264 s. There is also a clear inverse correlation between the test length and the performance of the wheel odometry, highlighting its drifting behavior. Overall the UWB localization system works well, with any combination of analyzed parameters of the EKF improving on the wheel odometry in all test runs, with only singular exceptions. Generally lower values for $\sigma^2_{anPos}$ improve the results, as in those cases the EKF puts a larger emphasis on the UWB measurements. But their trajectories are less smooth, than that of larger $\sigma^2_{anPos}$ values, and they are also more prone to large errors in the case of particularly poor anchor position measurements. This is indicated by the fact, that all exceptions to the better performance than the wheel odometry rule happen with low $\sigma^2_{anPos}$ values. The inclusion of inter-anchor

distance measurements and varying $\sigma^2_{dis}$ values, does not have a relevant impact on the systems results. The outage of the UWB measurements in Section 2 of Tests 4 to 6 mean, that the EKF is entirely reliant on the wheel odometry. As such the trajectories in those sections look identical to that of the wheel odometry. Also, the covariance of the EKF increases during these time periods, as there are no correcting measurements. After the outage is over, there is an immediate correction in the pose estimation of the robot, though not large enough, with the covariance also dropping again. The error in Section 3 is therefore reduced, but never reaches the magnitude of it in Section 1 again. Position estimation of the anchors in the multi-sectioned Tests 4 to 6 is much worse than in Tests 1 to 3 (RMSE of always more than 2.5 m compared to always less than 0.5 m). But the estimation is already poor in the beginning of the test run, which means the multiple sections are not the reason for the bad performance. The initial anchor position estimation though seems very important, as it does not change considerably over the course of the test run.

In total, we implement and demonstrate a UWB based localization approach, that works well, outperforming wheel odometry considerably. Also, it is capable to continue functioning in the case of an UWB node outage and self corrects after such an outage.

## 6.1 Future Work

Needless to say, a lot of work remains to be done. This comes down to four areas for improvement: First the embedded UWB system, preforming the distance measurements, second the evaluation approach, third the additional difficulties with UWB nodes in the 3D printed rockets, and finally overall system expansion.

As in this thesis we never calibrated the antenna delay for each node individually, this is to be done in the future. Using the setup as described in [75], with many nodes at once, seems promising. Also, accounting for range bias [76] will reduce the error further. In this implementation we always preform TWR between each pair of nodes, though the inter-tag distances are never used. Removing these from the full cycle of measurements reduces the latency slightly.

As in our tests $\sigma^2_{anPos} = 0.01$ produced less smooth but generally better results as $\sigma^2_{anPos} = 1$, values in between are to be analyzed. Also, instead of using the inter-anchor measurements as an observation of the EKF, where they do not have an impact, using them to determine the anchor positions together instead of independently of one another has to be studied. In the long term automatically aligning Riegl scans and using the result as a correction, will reduce the long term error of potentially longer test runs.

In this thesis the UWB anchors were always situated on tripods, instead of inside a plastic 3D printed rocket. The effect of the anchors inside a rocket on the ground has to be understood.

There are many potential ways to expand the system further. The UWB nodes can be used to preform localization in 3D/6Degrees of Freedom (DoF) instead of 2D/3DoF. Also, a larger network of anchors, without necessarily LOS between all of them, and potentially multiple robots, will increase the overall system size, which will require a system design adaptation, but has the opportunity for higher accuracy.

# Bibliography


[1] Andreas Nüchter, Lennart Werner, Martin Hesse, Dorit Borrmann, Thomas Walter, Montenegro Sergio, and Gernot Grömer. Uwb anchor based localization of a planetary rover. 2024.

[2] ESA. ESA-ESRIC Space Resources Challenge. `https://www.spaceresourceschallenge.esa.int/home`, 2021. Accessed: 24.10.2024.

[3] ÖWF. AMADEE-24 Mars simulation. `https://oewf.org/amadee-24/`, 2024. Accessed: 24.10.2024.

[4] Qorvo, Inc. DWM1000. `https://www.qorvo.com/products/p/DWM1000`, 2024. Accessed: 05.11.2024.

[5] Sergio Montenegro and Frank Dannemann. Rodos - real time kernel design for dependability. *DASIA 2009 - DAta Systems in Aerospace*, 669:66, 2009.

[6] Stanford Artificial Intelligence Laboratory et al. Robotic operating system, 2020.

[7] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

[8] Intel. Intel RealSense Tracking Camera T265. `https://www.intel.com/content/www/us/en/products/sku/192742/intel-realsense-tracking-camera-t265/specifications.html`, 2019. Accessed: 24.10.2024.

[9] Antoine Stephan. Resource allocation strategies and linear precoded ofdm optimization for ultra-wideband communications. 12 2008.

[10] Ieee standard for low-rate wireless networks. *IEEE Std 802.15.4-2020 (Revision of IEEE Std 802.15.4-2015)*, pages 1–800, 2020.

[11] Meilin Qian, Kai Zhao, Binghao Li, and Aruna Seneviratne. An overview of ultra-wideband technology and performance analysis of uwb-twr in simulation and real environment. *IPIN-WiP*, 2022.

[12] T.K.K. Tsang and M.N. El-Gamal. Ultra-wideband (uwb) communications systems: an overview. In *The 3rd International IEEE-NEWCAS Conference, 2005.*, pages 381–386, 2005.

[13] Walter Hirt. Ultra-wideband radio technology: overview and future research. *Computer Communications*, 26(1):46–52, 2003.

[14] Bundesnetzagentur. Allgemeinzuteilung von Frequenzen für die Nutzung durch Ultrabreitbandgeräte (UWB) Vfg. 135 / 2019. https://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/Sachgebiete/Telekommunikation/Unternehmen_Institutionen/Frequenzen/Allgemeinzuteilungen/FunkanlagenGeringerReichweite/2019Vfg135_UWB.pdf?__blob=publicationFile&v=2, 2019. Accessed: 30.10.2024.

[15] Federal Communications Commission. 47 CFR § 15.519 - Technical requirements for hand held UWB systems. https://www.govinfo.gov/content/pkg/CFR-2010-title47-vol1/pdf/CFR-2010-title47-vol1-sec15-517.pdf, 2002. Accessed: 30.10.2024.

[16] Moe Z Win, Davide Dardari, Andreas F Molisch, Werner Wiesbeck, and W Jinyun Zhang. History and applications of uwb. Institute of Electrical and Electronics Engineers, 2009.

[17] Eirini Karapistoli, Fotini-Niovi Pavlidou, Ioannis Gragopoulos, and Ioannis Tsetsinas. An overview of the ieee 802.15.4a standard. *IEEE Communications Magazine*, 48(1):47–53, 2010.

[18] P. Withington, R. Reinhardt, and R. Stanley. Preliminary results of an ultra-wideband (impulse) scanning receiver. In *MILCOM 1999. IEEE Military Communications. Conference Proceedings (Cat. No.99CH36341)*, volume 2, pages 1186–1190 vol.2, 1999.

[19] Decawave. *DW1000 User Manual*, 2017.

[20] Rakhesh Singh Kshetrimayum. An introduction to uwb communication systems. *IEEE Potentials*, 28(2):9–13, 2009.

[21] N. Fourty, T. Val, P. Fraisse, and J.-J. Mercier. Comparative analysis of new high data rate wireless communication technologies "from wi-fi to wimax". In *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services - (icas-isns'05)*, pages 66–66, 2005.

[22] Lochan Verma, Mohammad Fakharzadeh, and Sunghyun Choi. Wifi on steroids: 802.11ac and 802.11ad. *IEEE Wireless Communications*, 20(6):30–35, 2013.

[23] Department of Computer Science 8, University of Würzburg. SKITH - Skip The Harness. https://www.informatik.uni-wuerzburg.de/aerospaceinfo/forschung-und-entwicklung-prof-dr-sergio-montenegro/projekte/skith/, 2018. Accessed: 14.12.2024.

[24] Bashar Alsadik and Samer Karam. The simultaneous localization and mapping (slam)-an overview. *Journal of Applied Science and Technology Trends*, 2:120–131, 11 2021.

[25] Hugh Durrant-Whyte, David Rye, and Eduardo Nebot. Localization of autonomous guided vehicles. In Georges Giralt and Gerhard Hirzinger, editors, *Robotics Research*, pages 613–625, London, 1996. Springer London.

[26] Ernst D Dickmanns, Birger Mysliwetz, and Thomas Christians. An integrated spatio-temporal approach to automatic visual guidance of autonomous vehicles. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(6):1273–1284, 1990.

[27] Hamid Taheri and Zhao Chun Xia. Slam; definition and evolution. *Engineering Applications of Artificial Intelligence*, 97:104032, 2021.

[28] Udo Frese, René Wagner, and Thomas Röfer. A slam overview from a user's perspective. *KI-Künstliche Intelligenz*, 24:191–198, 2010.

[29] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[30] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[31] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[32] Weifeng Chen, Chengjun Zhou, Guangtao Shang, Xiyang Wang, Zhenxiong Li, Chonghui Xu, and Kai Hu. Slam overview: From single sensor to heterogeneous fusion. *Remote Sensing*, 14(23), 2022.

[33] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4:333–349, 1997.

[34] Jianbo Shi et al. Good features to track. In *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pages 593–600. IEEE, 1994.

[35] Mike Stephens and C Harris. A combined corner and edge detector. In *Alvey vision conference*, volume 15, 1988.

[36] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 430–443. Springer, 2006.

[37] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[38] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[39] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[40] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *2011 International Conference on Computer Vision*, pages 2320–2327, 2011.

[41] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[42] P. Newman and Kin Ho. Slam-loop closing with visually salient features. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 635–642, 2005.

[43] Ieee standard for information technology– local and metropolitan area networks– specific requirements– part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (wpans): Amendment 1: Add alternate phys. *IEEE Std 802.15.4a-2007 (Amendment to IEEE Std 802.15.4-2006)*, pages 1–210, 2007.

[44] Dries Neirynck, Eric Luk, and Michael McLaughlin. An alternative double-sided two-way ranging method. In *2016 13th Workshop on Positioning, Navigation and Communications (WPNC)*, pages 1–4, 2016.

[45] Maria Isabel Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43(46):3736–3741, 2004.

[46] D Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[47] Tomas Cipra and Rosario Romera. Kalman filter with outliers and missing observations. *TEST: An Official Journal of the Spanish Society of Statistics and Operations Research*, 6:379–395, 02 1997.

[48] Erik Smistad, Thomas Falch, Mohammadmehdi Bozorgi, Anne Elster, and Frank Lindseth. Medical image segmentation on gpus - a comprehensive review. *Medical Image Analysis*, 20:1–18, 02 2015.

[49] Automation Group (Jacobs University Bremen) and Knowledge-Based Systems Group (University of Osnabrück). 3dtk – the 3d toolkit. `http://slam6d.sourceforge.net/`, 2011. Accessed: 24.10.2024.

[50] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, 1987.

[51] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International Journal of Computer Vision*, 13:119–152, 1994.

[52] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and Vision Computing*, 10(3):145–155, 1992. Range Image Understanding.

[53] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001.

[54] Thomas Wisspeintner and Ansgar Bredenfeld. Volksbot - ein modularer roboterbaukasten für ausbildung und forschung. pages 168–172, 01 2005.

[55] Dorit Borrmann. volksbot. https://github.com/JMUWRobotics/volksbot, 2024.

[56] Logitech. Logitech Wireless Gamepad F710. https://www.logitechg.com/de-de/products/gamepads/f710-wireless-gamepad.940-000145.html?srsltid=AfmBOooAqbAHfHEbCS8iCzH--NAGvZHsgsHyh21dsKg5knNwMKSf7aKu, 2024. Accessed: 04.11.2024.

[57] SICK AG. SICK LMS100-10000. https://www.sick.com/be/de/produkte/lidar-und-radarsensoren/lidar-sensoren/lms1xx/lms100-10000/p/p109841, 2024. Accessed: 04.11.2024.

[58] RIEGL Laser Measurement Systems GmbH. RIEGL VZ-400. http://www.riegl.com/uploads/tx_pxpriegldownloads/10_DataSheet_VZ-400_2017-06-14.pdf, 2017. Accessed: 24.10.2024.

[59] Michael Strohmeier, Thomas Walter, Julian Rothe, and Sergio Montenegro. Ultra-wideband based pose estimation for small unmanned aerial vehicles. *IEEE Access*, 6:57526–57535, 2018.

[60] STMicroelectronics. STM32F407/417. https://www.st.com/en/microcontrollers-microprocessors/stm32f407-417.html, 2024. Accessed: 05.11.2024.

[61] Pablo Corbalán and Gian Picco. Ultra-wideband concurrent ranging. *ACM Transactions on Sensor Networks*, 16:1–41, 10 2020.

[62] Shashi Shah and Tanee Demeechai. Multiple simultaneous ranging in ir-uwb networks. *Sensors*, 19(24), 2019.

[63] Bernhard Großwindhager, Carlo Alberto Boano, Michael Rath, and Kay Römer. Concurrent ranging with ultra-wideband radios: From experimental evidence to a practical solution. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1460–1467, 2018.

[64] Francesco Martinelli and Fabrizio Romanelli. A slam algorithm based on range and bearing estimation of passive uhf-rfid tags. In *2021 IEEE International Conference on RFID Technology and Applications (RFID-TA)*, pages 20–23, 2021.

[65] Emidio DiGiampaolo and Francesco Martinelli. Range and bearing estimation of an uhf-rfid tag using the phase of the backscattered signal. *IEEE Journal of Radio Frequency Identification*, 4(4):332–342, 2020.

[66] Johannes Traa and Paris Smaragdis. A wrapped kalman filter for azimuthal speaker tracking. *IEEE Signal Processing Letters*, 20(12):1257–1260, 2013.

[67] Ivan Marković, Josip Ćesić, and Ivan Petrovic. On wrapping the kalman filter and estimating with the SO(2) group. 08 2017.

[68] Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160, 2011.

[69] Stefan Kohlbrecher and Johannes Meyer. hector_slam. `https://wiki.ros.org/hector_slam`, 2024.

[70] Bruno Steux and Oussama El Hamzaoui. tinyslam: A slam algorithm in less than 200 lines c-language program. In *2010 11th International Conference on Control Automation Robotics & Vision*, pages 1975–1979, 2010.

[71] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[72] Kristóf Attila Horváth, Gergely Ill, and Ákos Milánkovich. Calibration method of antenna delays for uwb-based localization systems. In *2017 IEEE 17th International Conference on Ubiquitous Wireless Broadband (ICUWB)*, pages 1–5, 2017.

[73] Shashi Shah, Krit Chaiwong, La-Or Kovavisaruch, Kamol Kaemarungsi, and Tanee Demeechai. Antenna delay calibration of uwb nodes. *IEEE Access*, 9:63294–63305, 2021.

[74] Xinzhe Gui, Shuli Guo, Qiming Chen, and Lina Han. A new calibration method of uwb antenna delay based on the ads-twr. In *2018 37th Chinese Control Conference (CCC)*, pages 7364–7369, 2018.

[75] Decawave. *APS014 Application Note, Antenna Delay Calibration of DW1000-based Products and Systems*, 2024.

[76] Decawave. *APS011 Application Note, Sources of Error in DW1000-based Two-Way-Ranging (TWR) Schemes*, 2024.

# Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Würzburg, December 2024